Komputerowe modelowanie zjawisk rynkowych Documentation

Wydanie 0.1

Łukasz Machura

December 16 2012

Spis treści

Zajęcia dla 1 roku studiów magisterskich ekonofizyki na Uniwersytecie Śląskim w semestrze zimowym roku akademickiego 2012/2013.

Autor Łukasz Machura, ZFT, Uniwersytet Śląski

Wersja 0.1 (Ekonofizyka UŚ, semestr zimowy 2012)

PDF KMZR

ROZDZIAŁ 1

Wstęp

Wymagania teoretyczne

- 13. Łukaszewski, J. Sładkowski, Wybrane zagadnienia analizy rynków finansowych
- 13. Łukaszewski, J. Sładkowski, Wprowadzenie do funkcjonowania rynków finansowych
- 10. Łuczka, Procesy i zjawiska losowe

Wymagania techniczne

- podstawy programowania (Sage/Python lub Matlab/Octave lub C/C++ lub cokolwiek innego...)

Uwagi

- Jako, że zajmujemy sie metodami komputerowymi, warto ten podręcznik czytać przy komputerze, jednocześnie powtarzając prezentowane obliczenia samodzielnie. Większość zaprezentowanego kodu dostępna jest w projekcie kmzj, zalecam jednak wpisywanie większości algorytmów samodzielnie. Nauka poprzez palce jest bardzo ważna, pozwala nabyć niezbędnych umiejętności pracy przy (na?) komputerze. Zwykłe "copy-paste" nie nauczy nikogo wiele. Dodatkowo zachęcam do dowolnych zmian w kodach prezentowanych w tym kursie - do optymalizacji, skracania, przyspieszania itp. Nie zawsze pisałem ten kod z myślą o zastosowaniach końcowych. W większości przypaków ma on być szkoleniowy i intuicyjny (jeżeli dla kogoś Python jest intuicyjny).
- W tym kursie do numerycznych realizacji używać będziemy pakietu Sage. Jest to dostępny dla każdego program typu open-source bazujacy na języku Python.

Liczby Losowe

2.1 Liczby losowe i pseudolosowe

Intuicyjnie dość dobrze rozumiemy co oznacza termin *liczba losowa*. Każdy z nas choć raz w życiu podrzucił monetę do góry po to, by "ślepy los" zdecydował za niego o jakimś wyborze (jeżeli w ten sposób zdecydowaliście o wyborze studiów, to szczerze mówiąc - gratuluję). Oczywiście na monecie nie ma żadnych liczb, ale można sobie potraktować reszkę (R) jako 0 a orła (O) jako 1 (co bardzo dobrze reprezentuje fałsz i prawdę lub niemożliwe i pewne zdarzenie w teorii prawdopodobieństwa). Teraz już możemy sobie podrzucać monetę i na kartce papieru zapisywać kolejne wylosowane (wyrzucone) przez nas liczby

```
0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1,
```

co odpowiada oczywiście wyrzuceniu kolejno

```
R, O, R, R, R, R, O, R, O, O, R, R, O, R, O.
```

W naszym przypadku zapiszemy sobie te liczby od razu do listy w notatniku Sage.

```
rzuty = [0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1]
```

Mamy teraz je dostępne pod zmienną `rzuty`. Do prostych zagadnień, gdzie potrzebne jest nam kilka, czy nawet kilkanaście takich liczb, bez problemu możemy poradzić sobie rzucając monetą. Jeżeli potrzebujemy zadecydować o wyborze pomiędzy trzema możliwościami możemy użyć sześciennej kości do gry i przykładowo wybrać wynik poprzez działanie modulo 3 (mod3). Tym razem dostaniemy trzy możliwe liczby `0, 1, 2`

```
# rzuty kością
5, 3, 6, 5, 6, 6, 5, 3, 2, 4, 3, 1, 1, 6, 1,

# modulo 3
2, 0, 0, 2, 0, 0, 2, 0, 2, 1, 0, 1, 1, 0, 1.
```

Sytuacja zrobi się jednak nieco bardziej skomplikowana, gdy będziemy potrzebować tysiąc, milion czy bilion takich liczb. Jeżeli nawet grupa 10 studentów była by w stanie wyrzucić monetą tysiąc losowych zer i jedynek w pół godziny (włączając w to zapisywanie w liście Sage lub nawet na kartce papieru) to uzyskanie miliona liczb jest praktycznie nie do zrobienia w ten sposób. problem pojawia się też w momencie, gdy chcielibyśmy mieć liczby naturalne z zakresu np: $0 - 10$, czy w końcu losowe liczby zmiennoprzecinkowe. Metody chałupnicze w tym momencie się kończą.

Z pomocą może przyjść nam komputer (w końcu ten wykład jest o metodach komputerowych). Obecnie znakomita większość języków programowania (przynajmniej tych realnie wykorzystywanych [1]) posiada w swoich standardo-

[1] Generator liczb pseudolosowych można napisać nawet dla tak egzotycznych języków jak Brainf*ck.

wych bibliotekach funkcje (metody, klasy) umożliwiające wygenerowanie liczby (pseudo)losowej z przedziału `[0,1)` lub też `[0,RAND_MAX]`, gdzie ów `RAND_MAX` to stała zależna od architektury komputera, kompilatora i bibliotek.

W Sage liczby losowe uzyskuje się poprzez funkcję `random()`. Zwraca ona liczbę losową z przedziału `[0.0, 1.0)`. Wykorzysując proste wyrażenie listowe możemy przypisać do listy N liczb losowych.

```
N = 1000
lista = [random() for i in xrange(N)]
```

Inna funkcja `randint(a, b)`, zwraca liczby całkowite z przedziału `[a,b]`. Czyli symulacja rzutu monetą może być zrealizowana poprzez

```
rzut_moneta = [randint(0,1) for i in xrange(N)]
```

Zadanie 2.2.1 Zamodeluj w Sage rzut kością. Wygeneruj listę 1000 liczb odzwierciedlających 1000 rzutów symetryczną sześcienną kością do gry. Wynik zapisz w zmiennej `rzut_kostka`.

Matematycznie rzecz biorąc liczbę losową można utożsamić z wartością jaką przybiera pewna zmienna losowa ξ. Możemy napisać, że dla procesu jakim jest rzut kością zmienna losowa ξ może przybierać wartości `0` lub `1`. Matematyczne konsekwencje poznaliście już na wykładzie Procesy i zjawiska losowe, tutaj zajmiemy się znacznie szerzej generowaniem liczb losowych i wykorzystaniem ich właśnie do realizacji procesów losowych, ze szczególnym uwzględnieniem zastosowania dla rynków finansowych, czy w ogólności w modelach ekonomicznych.

No koniec tego rozdziału musimy sobie powiedzieć jasno: program komputerowy bazujący na deterministycznym generatorze liczb losowych może wygenerować tylko i wyłącznie liczby pseudolosowe, czyli takie, które tylko imitują prawdziwe liczby czysto losowe. Te ostatnie osiągalne są tylko procesie rzeczywistym. Możemy jednak za pomocą takich generatorów uzyskać ciąg liczb (bitów), który pod pewnymi względami będzie nierozróżnialny od ciągu uzyskanego z prawdziwie losowego źródła (np: z rzutu rzeczywistą kością).

2.2 Generatory liczb

Generator liczb losowych (RNG, z ang. random number generator) lub nieco bardziej ściśle *generator zdarzeń losowych* (REG, z ang. random event generator) to układ produkujący losowy ciąg elementów binarnych (bitów) najczęściej ułożony w postaci szeregu liczb losowych. Z punktu widzenia sposobu generowania liczb losowych wyróżniamy generatory sprzętowe (fizyczne, rzeczywiste) i programowe.

2.2.1 Generatory sprzętowe

TRNG (z ang. True RNG) - działające na zasadzie obrazowania właściwości i parametrów fizycznego procesu stochastycznego. Może to być ów rzut kością, monetą, wybieranie karty z talii kart itp. Wykorzystywać można też: efekt fotoelektryczny, szum termiczny, szum śrutowy, proces zaniku radioaktywnego...

2.2.2 Generatory programowe

PRNG, (z ang. Pseudo RNG) - działające na zasadzie deterministycznego obliczania ciągu liczb, które wyglądają jak liczby losowe. Algorytmy realizujące PRNG istnieją już ponad pół wieku i są obecnie zaimplementowane dla większości języków programowania. Na podstawie początkowej wartości nazywanej ziarnem czy zarodkiem (z ang. seed) oblicza kolejne wartości. Obie prezentowane funkcje Sage (`random` i `randint`) korzystają właśnie z jednego z takich algorytmów, zwanego Mersenne Twister. Jest to obecnie chyba najbardziej popularny algorytm opracowany w 1997 roku. Np. Matlab/GNU Octave też wykorzystuje ten algorytm. Jest on stosunkowo skomplikowany i może być trudny do realizacji, dlatego też omówimy sobie dużo prostszy, liniowy generator i omówimy jego zalety i (przede wszystkim) wady.

Programowe generowanie liczb losowych [2] oparte jest na rekurencji

$$x_i = f(x_{i-1}, x_{i-2}, \ldots, x_{i-k}),$$

czy w nieco bardziej zwartej formie

$$x_i = f(x_{i-1}).$$

Sekwencje te będą w oczywisty sposób deterministyczne. Problem polega na wygenerowaniu liczb których własności bardzo dobrze przypominają główne własności liczb prawdziwie losowych. Dodatkowo sekwencje liczb pseudolosowych będą powtarzały się co pewien okres, więc dość istotne jest aby generator takich liczb posiadał ów okres jak najdłuższy.

Liniowy generator kongruencyjny

LCG (linear congruential generator) wyznaczony jest przez metodę rekurencyjną

$$X_{n+1} = (aX_n + c) \mod m.$$

Stan początkowy to wartość ziarna (zalążka). Nie jest on zbytnio bezpieczny - istnieją techniki identyfikacji parametrów modelu na podstawie obserwacji wyników. Dla niektórych parametrów jest prawie losowy a dla innych dość szybko staje się okresowy. W powyższej definicji x_0 to ziarno (zalążek), a mnożnik, c przesunięcie a $m \in \mathbb{Z}$ nazywamy modułem. Dwie liczby nazywamy kongruentnymi (przystającymi) modulo m jeżeli ich różnica jest podzielna przez m. Jeżeli $0 \leq a < m$ oraz $a \equiv b \mod m$ wtedy a nazywamy resztą $b \mod m$. Liczbę a można łatwo obliczyć z

$$a = b - \lfloor b/m \rfloor \times m$$

gdzie funkcja podłoga (z ang. floor) $\lfloor \circ \rfloor$ oblicza największą liczbą całkowitą mniejszą od $\circ$.

Jeżeli weźmiemy $c = 0$ dostaniemy multiplikatywny generator kongruencyjny. Jeżeli chodzi o moduł, to typowymi wartościami będą potęgi 2^k, a wartościami tych potęg bedą typowe wielkości maszynowe dla przechowywania liczb całkowitych. Tak było przynajmniej dla wczesnych realizacji takiego generatora, co związane było z możliwością łatwej redukcji modulo poprzez wykorzystanie przepełnienia w stałopozycyjnej reprezentacji liczb w operacji mnożenia (w ciele liczb całkowitych) ax_i. W operacjach stałoprzecinkowych pierwszy bit reprezentuje znak, wobec czego w wyniku takiego mnożenia zamiast liczb z zakresu $[0, 2^{32} - 1)$ dostaniemy liczby z zakresu $[-2^{31} + 1, 2^{31} - 1]$. W ogólności wykonując operacje na liczbach większych od $2^{31} - 1$ jako wynik zachowujemy tylko bity niskiego rzędu.

Mnożnik a wybierać trzeba w taki sposób, aby LCG miał jak najdłuższy okres. Na 32-bitowych maszynach popularnymi wartościami początkowo były $m = 2^{32}$ i $a = 65539$. Jako, że dzisiejsze komputery są na tyle wydajne, by przeprowadzać reducję modulo bardzo wydajnie, wiele ówczesnych implementacji generatora wykorzystuje operacje zmiennoprzecinkowe o zwiększonej precyzji. Inne wartości $a = 1099087573, 2396548189, 3934873077, 2304580733$ również produkują porządne sekwencje liczb losowych.

Innym dobrym wyborem dla *m* jest podstawienie dużej liczby pierwszej *p*. Wtedy okresem LCG będzie *p-1* jeżeli tylko mnożnik ustawimy jako jego pierwiastek pierwotny. Szczególnie ważne wydają się być liczby pierwsze postaci $2^p - 1$, nazywane liczbami Mersenne'a. Na maszynach 32-bitowych popularnym wyborem bywa para $m = 2^{31} - 1$ i jej pierwiastek pierwotny $a = 7^5 = 16807$.

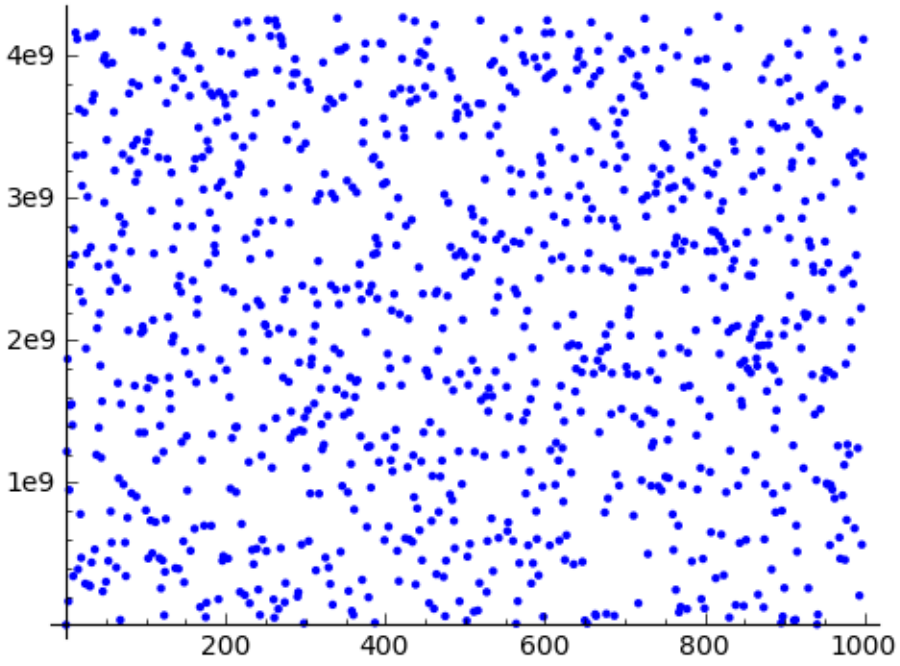
Implementacja LCG w Sage nie powinna nastręczać zbytnich problemów:

```
def myLCG(x, a=1664525, b=1013904223, m=2**32):
  return mod(a*x+b,m)
```

Możemy teraz wygenerować N liczb używając LCG i zmagazynowac je w pythonowskiej liście.

[2] Od tej chwili będziemy zawsze pisać *liczba losowa* a mieć na myśli *liczbę pseudolosową*, chyba, że napisane zostanie explicite, że mówimy o rzeczywistych liczbach losowych.

```
def get_from_LCG(n=1, seed=123):
  ret = [seed]
  for i in xrange(n-1):
    ret.append(myLCG(ret[i]))
  return ret

lcg_list = get_from_LCG(N)
```



Rysunek 2.1: 1000 liczb losowych wygenerowanych generatorem liniowym LCG

Jak widać, program generuje liczby losowe z zakresu [0,m).

Rejestr przesuwający z liniowym sprzężeniem zwrotnym

TBA

Generator Wichmanna – Hilla

TBA

Mersenne Twister

TBA

W dalszej części wykładu (a raczej ćwiczeń) będziemy bazować na domyślnym generowaniu liczb losowych w Sage. Posłuży nam do tego wspominana już funkcja `random()` zwracająca liczbę pseudolosową o rozkładzie jednorodnym na odcinku [0,1) (co często oznaczane jest poprzez $U(0,1)$).

$$U(0,1) = \begin{cases} 1 & 0 \leq x < 1 \\ 0 & \text{poza} \end{cases}$$

Aby uzyskać liczbę z przedziału [0,12.76) wystarczy po prostu pomnożyć liczbę zwracną przez `random()` przez prawą granicę

```
random()*12.76
```

a żeby uzyskać listę 123 liczb z przedziału [-13.3, 33.1) należy wykonać

```
[random(33.1+13.3) - 13.3 for i in xrange(123)]
```

W ogólności do wygenerowania listy N liczb losowych z przedziału [A,B) należy użyć polecenia

```
N = 100
A = -10
B = 20
[random(B-A) + A for i in xrange(N)]
```

Zadanie 2.2.2 Zmodyfikuj definicję `mylcg` tak, aby funkcja zwracała liczby losowe z przedziału [0,1).

Zadanie 2.2.3 LCG zdefiniowany tak jak powyżej produkuje stosunkowo dobre liczby losowe (prace naukowe nad tym stosunkowo prostym generatorem trwają do dzisiaj, dowodzone są coraz to inne okresy bazujące na wyborze różnych zestawów parametrów a, c, m). Naszym zadaniem będzie natomiast zepsucie takiego generatora. Proszę znaleźć (numerycznie) 4 zestawy parametrów definiujących LCG takich, aby okres generatora był krótki. Wykreśl w Sage 4 rysunki LCG(N) (dla powiedzmy N=1000) dla owych parametrów. Powinieneś zauważyć regularność.

2.3 Generowanie liczb losowych o zadanym rozkładzie

Jako, że już dysponujemy generatorem liczb losowych o rozkładzie jednostajnym na odcinku $[0, 1) - U(0, 1)$ to możemy pokusić się o wygenerowanie liczb losowych o różnych innych rozkładach prawdopodobieństwa. Znane jest kilka metod generowania takich liczb. Wszystkie przedstawione tutaj będą opierały się na tym, że umiemy generować liczby z rozkładem U(0,1). Szczególną uwagę poświęcimy generowaniu liczb z rozkładem N(0,1). Jest to standardowy zapis oznaczający rozkład normalny (Gaussa) o średniej równej 0 i odchyleniu standardowym równym 1. Zanim omówimy pierwszą metodę, wcześniej zdefiniujemy sobie pojęcie *histogramu*. Będzie nam on potrzebny do wizualizacji rozkładów (czy raczej ich gęstości) z wygenerowanych liczb losowych.

2.3.1 Histogram

Wikipedia definiuje histogram jako jeden z graficznych sposobów przedstawiania rozkładu empirycznego cechy. Konstruuje się go jako szereg prostokątów odpowiadających liczebności elementów wpadających do określonego przedziału klasowego. Szerokości przedziałów klasowych mogą mieć stałe lub zmienne długości. W bardziej matematycznym sensie histogram to funkcja zliczająca ilość obserwacji pasujących do oddzielnych przedziałów klasowych. Jeżeli *n* oznacza liczbę wszystkich obserwacji, a *k* to liczba przedziałów, wtedy histogram m_i spełnia następujący warunek

$$n = \sum_{i=1}^{k} m_i$$

Ideę histogramu najlepiej zrozumiec na przykładzie. Mamy następującą listę liczb

```
l = [1, -3, -5, -1, -3, 1, 5, 1, 3, -3, 4, 2, 4, -1, 4, 5, -2, 4, 3, -4]
```

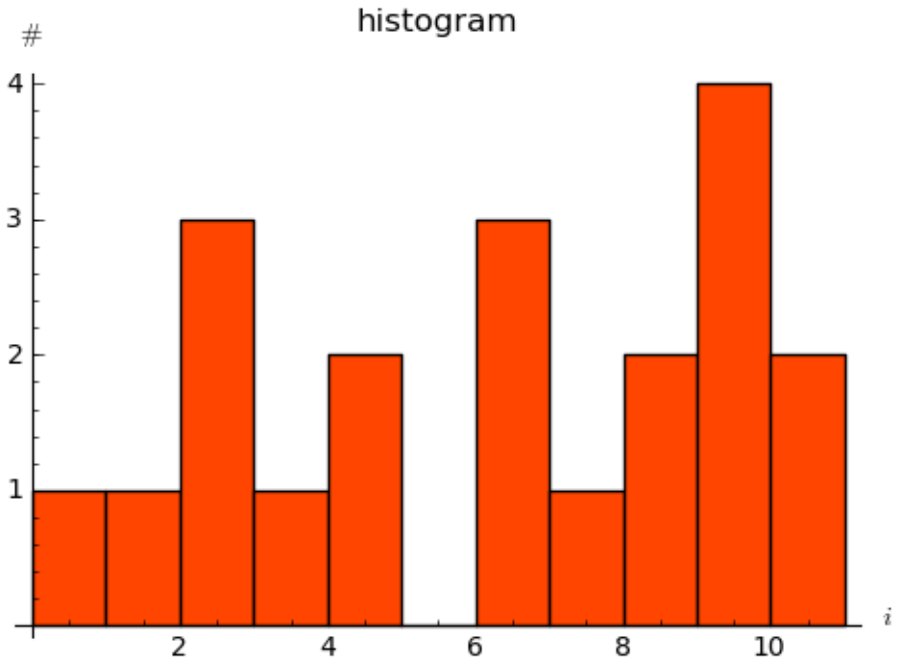
Budując histogram na początku musimy ustalić szerokość przedziału. Zacznijmy od łatwiejszej wersji: niech szerokość będzie stała. Najlepiej podzielić ową listę na przedziały zawierające liczby całkowite. W zasadzie wystarczy zliczać ile jest poszczególnych liczb całkowitych w liście l. Zróbmy to. Widzimy, że mamy

-5	-4	-3	-2	-1	0	1	2	3	4	5
1	1	3	1	2	0	3	1	2	4	2

W zasadzie mamy już nasz histogram. Jeżeli posumujemy ilość elementów listy (`len(l)`), oraz obliczymy n zobaczymy, że dostaniemy tą samą liczbę (=20). Pozostaje narysować ów histogram. Na odciętej musimy odłożyć przedziały klasowe a na rzędnej liczebności danego przedziału. Przyjęło się rysować histogram używając słupków. Sage na chwilę obecną posiada kilka metod narysowania takiego histogramu. Jeżeli nie zależy nam na poprawnym opisaniu odciętej (np: chcemy tylko zobaczyć kształt histogramu), wystarczy napisać

```
h = [l.count(i) for i in range(-5,6)]
bar_chart(h, width=1, color="orangered").show(axes_labels=['$i$','$\#$'], title="histogram")
```

Co pozwoli nam wygenerować taki rysunek:



Rysunek 2.2: Prosty wykres liczebności, gdzie wykorzystaliśmy funkcję box_plot().

Nie jest to prawdziwy histogram, bowiem odłożone na osi OY liczebności powinny odpowiadać rzeczywistym wartościom (przedziałom). Możemy skorzystać z pakietu `Time Series` dostępnego w Sage. Wystarczą prosta komenda aby uzyskać dostęp do wielu statystycznych funkcji typowych dla analizy szeregu czasowego.

```
v = finance.TimeSeries(l)
```

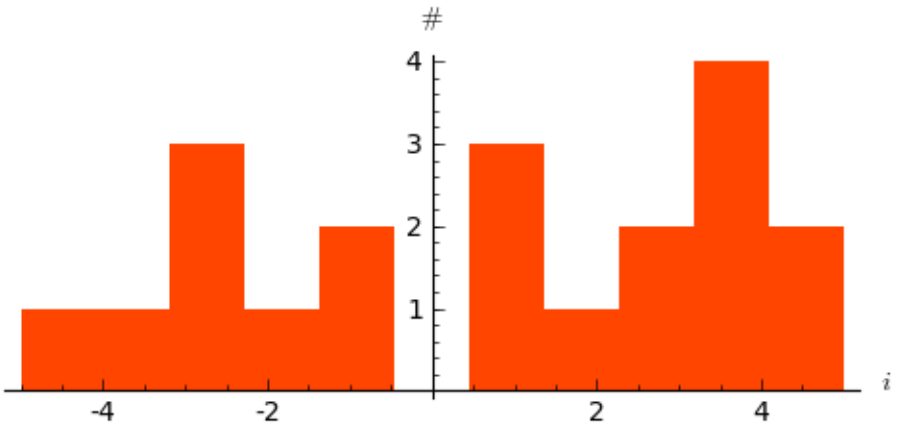
I teraz aby obliczyć histogram dla 10 równych przedziałów (od minimalnej do maksymalnej wartości występującej w liście `l`), wystarczy napisać

```
v.histogram(bins=11)
```

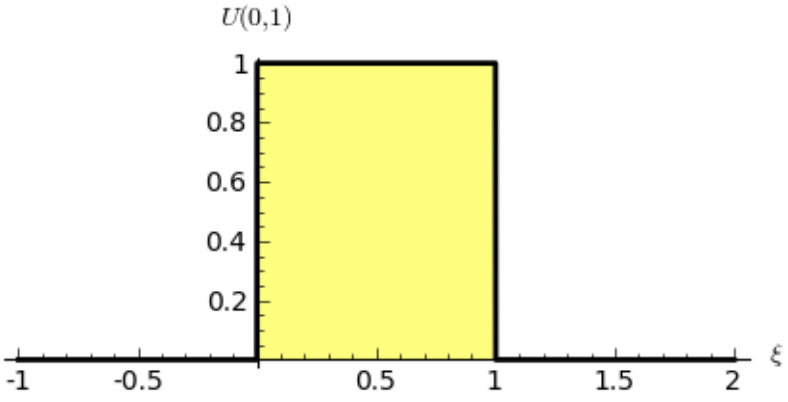
a aby narysować jego wykres

```
v.plot_histogram(bins=11, normalize=0, color="orangered", axes_labels=['$i$','$\#$'])
```

Oczywiście całą procedurę można powtórzyć dla liczb zmiennoprzecinkowych (rzeczywistych, wymiernych). W tym wypadku należałoby oczywiście policzyć ile posiadanych liczb wpada do zdefiniowanych "pudełek". Zobaczmy drugi przykład, gdzie obliczymy i narysujemy w Sage histogram dla dziesięciu tysięcy liczb z U(0,1). Powinniśmy dostać



Rysunek 2.3: Historgam dla listy `l` uzyskany z wykorzystaniem pakietu `TimeSeries`

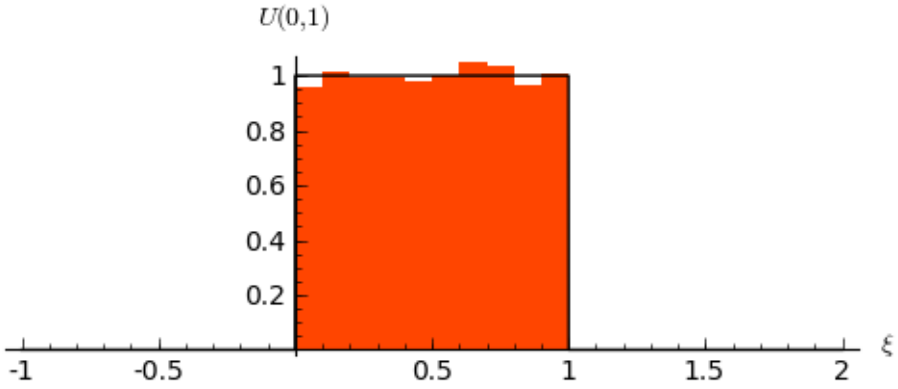


Rysunek 2.4: Wykres rozkładu U(0,1)

Przykład 2 Wygenerujemy 10000 liczb losowych a następnie dla nich obliczymy i narysujemy histogram.

```
N = 10000
u01 = [random() for i in xrange(N)]
fu01 = lambda x: 0 if x < 0 or x > 1 else 1
v = finance.TimeSeries([random() for i in xrange(N)])
plot1 = plot(fu01,(-1,2), thickness=1, color="black")
plot2 = v.plot_histogram(bins=10, color="orangered")
(plot1 + plot2).show(axes_labels=[r'$\xi$',r'$U(0,1)$'])
```

Ostatnia linia wyrysuje nam obie funkcje na jednym wykresie. Zachęcam czytelnika do poeksperymentowania z powyższym kodem - można zmienić liczbę prób *N* i łatwo zobaczyć jak histogram zaczyna oddalać się od teoretycznego rozkładu dla małych N i jak zbliża się dla dużych. Można też zobaczyć jak ilość przedziałów (parametr `bins`) wpływa na otrzymany histogram.



Rysunek 2.5: Wykres rozkładu U(0,1) + histogram miliona prób.

2.3.2 Metoda inwersyjna

Każdy rozkład prawdopodobieństwa może być jednoznacznie scharakteryzowany poprzez pewną funkcję rzeczywistą zwaną **dystrybuantą**.

Dystrybuanta Niech $\mathbb{P}$ będzie rozkładem prawdopodobieństwa. Funkcję $\mathbb{F} : \mathbb{R} \to \mathbb{R}$ daną wzorem

$$\mathbb{F}(\xi) = \mathbb{P}((-\infty, \xi))$$

nazywamy dystrybuantą rozkładu $\mathbb{P}$.

W metodzie inwersyjnej żądany rozkład o dystrybuancie $\mathbb{F}$ uzyskuje się poprzez przekształcenie zmiennej losowej o rozkładzie U(0,1) za pomocą funkcji odwrotnej do $\mathbb{F}$.

Twierdzenie Załóżmy, że dystrybuanta $\mathbb{F}$ jest ściśle rosnąca. Jeśli zmienna losowa *u* ma rozkład *U(0,1)* to $\mathbb{F}^{-1}(u)$ ma dystrybuantę $\mathbb{F}$.

Dowód TBA

Algorytm wykorzystujący powyższe twierdzenie jest bardzo prosty i wygląda następująco:

1. Generujemy liczbę $u \in U(0.1)$.
2. Przekształcamy *u* stosując

   $x = \mathbb{F}^{-1}(u)$

   Wynikowa liczba losowa x posiada żądany rozkład $\mathbb{P}$.

Oczywiście skuteczność tej metody zależy bezpośrednio od tego czy możemy łatwo obliczyć $\mathbb{F}^{-1}$. Jeżeli tak - jest to najprostsza znana metoda generowania liczb losowych z danym rozkładem. Do rozkładów, do których można zastosować tą metodę należą wszystkie rozkłady, których dystrybuanta znana jest jawnie *oraz* można ją łatwo odwrócić. O takich rozkładach powiemy sobie niżej.

Rozkład wykładniczy

Przejdźmy wreszcie do generowania liczb losowych z rozkładem innym niż *U(0,1)*. Na początek weźmy jeden z najbardziej powszechnych, czy popularnych rozkładów prawdopodobieństwa - **rozkład wykładniczy**. Gęstość takiego rozkładu dana jest wzorem

$$f(\xi) = \lambda e^{-\lambda \xi}$$

Jak łatwo policzyć, dystrybuanta *F(x)* wynosi

$$F(x) = \int_{-\infty}^{x} f(\xi) d\xi = -e^{-\lambda \xi}\Big|_{-\infty}^{x} = -e^{-\lambda x} + 1,$$

a jej odwrotność

$$F^{-1}(u) = -\frac{1}{\lambda}\ln(1-u).$$

Spróbujmy wygenerować 5000 liczb o rozkładzie wykładniczym. Następnie obliczymy sobie histogram, unormujemy go i porównamy z teoretycznym rozkładem dla kilku wartości $\lambda = 0.5, 1, 1.5$.

```
f(xi, a) = a * exp(-a * xi)
F(u, a) = -log(1-u)/a
N = 5000
kolor = ["red", "green", "blue"]
parlist = [1.5, 1, 0.5]
p = []
for par in parlist:
  lista = [F(random(), par) for i in xrange(N)]
  v = finance.TimeSeries(lista)
  P = v.plot_histogram(bins=100, color=kolor[parlist.index(par)], alpha=0.5)
  P.set_aspect_ratio("automatic")
  p.append(P)
  p.append(plot(f(xi,par), 0, max(lista), thickness=2, color=kolor[parlist.index(par)],
  legend_label=r'$\lambda = %.1f$'%par))
sum(p).show(xmin=0, xmax=5, figsize=5, axes_labels=[r'$\xi$',r'$\lambda e^{-\lambda \xi}$'], fontsize
```

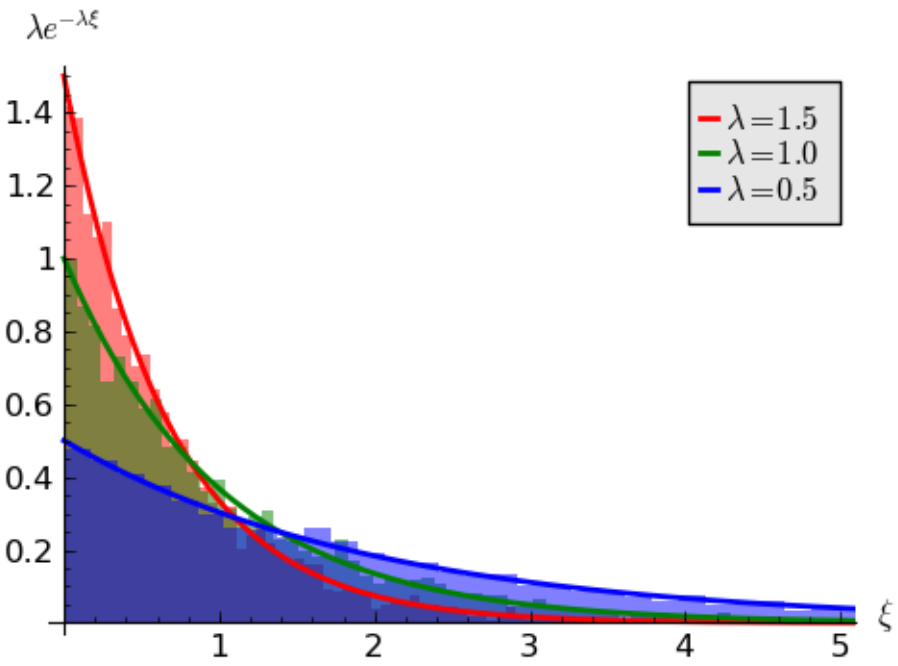
Jak widać na rysunku liczby losowe przekształcone metodą inwersji w oparciu o odwrotność dystrybuanty, dość dobrze odwzorowują rozkład wykładniczy. Lepszy wynik można oczywiście uzyskać zwiększając parametry `N` oraz `bins`.

Alternatywnie można wykorzystać przekształcenie bazujące na spostrzeżeniu, że liczby $1-u$ oraz $u(u \in U(0,1)$ posiadają ten sam rozkład *U(0,1)*.

Rozkład Cauchy'ego

Rozkład ten dany jest wzorem

$$f(\xi) = \frac{\sigma}{\pi(\xi^2 + \sigma^2)}$$



Rysunek 2.6: Wykres gęstości rozkładu wykładniczego oraz histogram z 5000 prób liczb losowych dla trzech wartości parametru λ.

Odwrotność dystrybuanty powyższego rozkładu wynosi

$$F^{-1}(u) = \sigma \tan\left[\pi\Big(u - \frac{1}{2}\Big)\right].$$

Stosując proste i bardzo naturalne przekształcenie oryginalnej zmiennej $v = 1/2 - u$ dostajemy nieco prostsze wyrażenie

$$F^{-1}(v) = \sigma \tan(\pi v).$$

Stosując podobne metody jak w poprzednim rozdziale możemy sprawdzić, czy powyższe przekształcenie generuje liczby z odpowiednim rozkładem.

Rozkład logistyczny

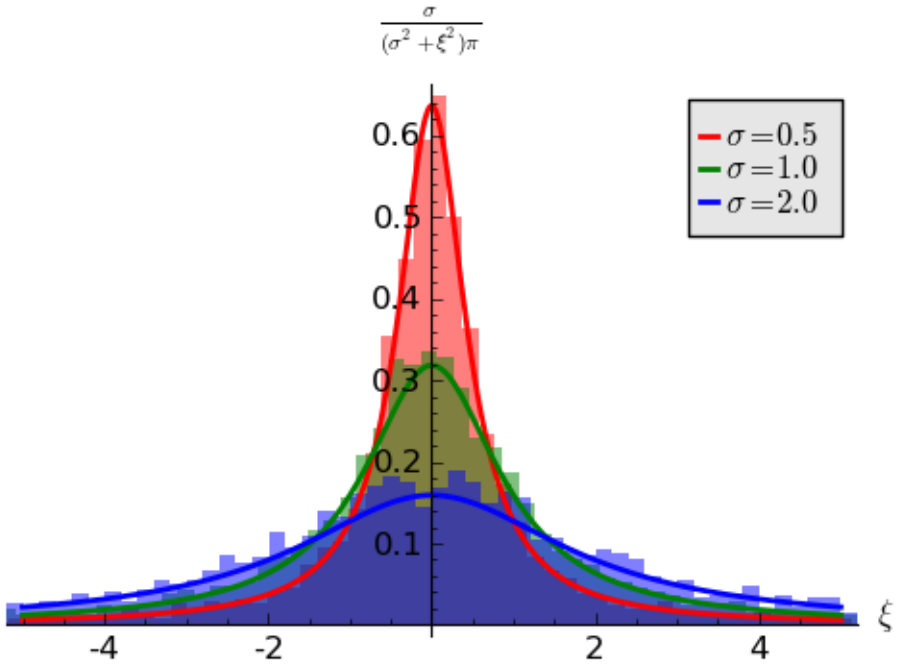
Rozkład ten dany jest wzorem

$$f(\xi) = \frac{1}{2 + e^{\xi} + e^{-\xi}}$$

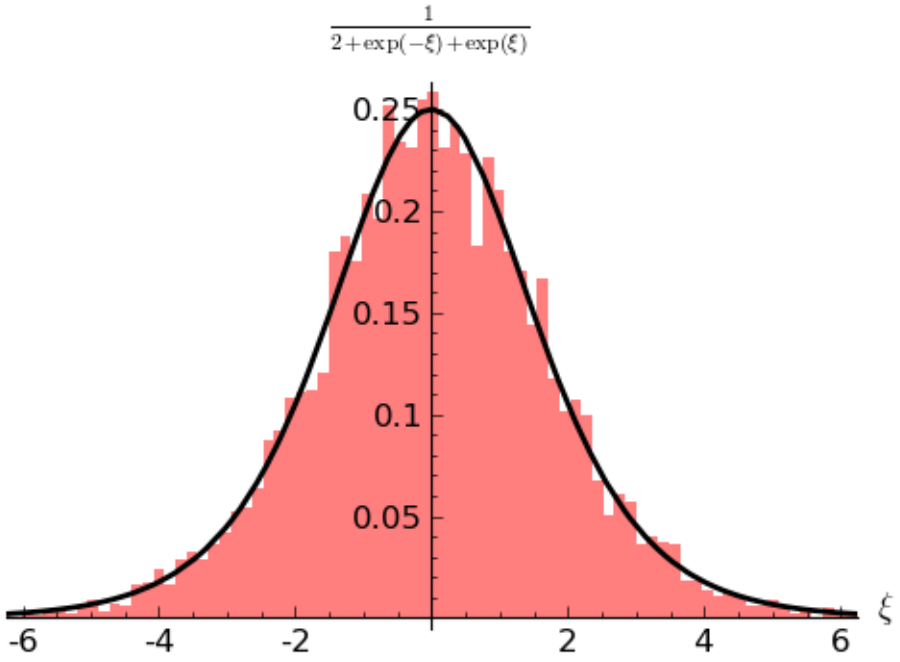
Odwrotność dystrybuanty powyższego rozkładu wynosi

$$F^{-1}(u) = \ln \frac{u}{1-u}.$$

Stosując podobne metody jak w poprzednim rozdziale możemy sprawdzić, czy powyższe przekształcenie generuje liczby z odpowiednim rozkładem.



Rysunek 2.7: Wykres gęstości rozkładu Cauchy'ego oraz histogram z 5000 prób liczb losowych dla trzech wartości parametru $\sigma = 0.5, 1, 2$.



Rysunek 2.8: Wykres gęstości rozkładu logistycznego oraz histogram z 200000 prób liczb losowych.

Zadanie 2.3.1 Wygeneruj 200000 liczb losowych z rozkładem Pareto. Narysuj ich unormowany histogram oraz funkcję gęstości. Porównaj obie funkcje zmieniając wszystkie parametry rozkładu.

Zadanie 2.3.2. Wygeneruj 1000 liczb losowych z rozkładem trójkątnym. Narysuj ich unormowany histogram oraz funkcję gęstości. Porównaj obie funkcje zmieniając wszystkie parametry rozkładu.

2.3.3 Metoda odrzucania

Polega ona na generowaniu liczb losowych o pewnym łatwo uzyskiwalnym rozkładzie, a następnie odrzuca się niektóre z nich. Pozostałe realizacje (liczby) posiadają żądany rozkład. Podstawą metody odrzucania jest twierdzenie

Twierdzenie Załóżmy, że zmienna X ma gęstość f oraz zmienna U ma rozkład równomierny na odcinku $[0, 1)$. Wówczas, dla dowolnego $c > 0$ para (X, Y) gdzie $Y \in cUf(X)$ ma rozkład równomierny na zbiorze $A = \{(x, y) : -\infty < x < \infty, 0 \leq y \leq cf(x)\}$. I na odwrót - jeżeli (X, Y) ma rozkład równomierny na zbiorze A, to X ma gęstość f.

Dowód TBA

Gęstość f jest zadana. Zakładamy, że istnieje inna gęstość g i dodatnia stała c (w rzeczywistości można albo obrać ją dość dowolnie, bądź lepiej obliczyć korzystając z takich bądź innych przesłanek) takie, że $f(x) \leq cg(x)$ dla wszystkich (!) $x \in \mathbb{R}$. Jak już założyliśmy na początku, rozkład g winien być łatwo uzyskiwalny (np: metodą inwersji).

Algorytm

A. Generujemy zmienną losową o rozkładzie równomiernym na zbiorze A zawartym pomiędzy osią *OX* a krzywą $y = xg(x)$. Można to zrobić tak jak w powyższym twierdzeniu

1. generujemy zmienną losową X o znanej (założonej przez nas, łatwej do uzyskania) gęstości g
2. generujemy $U \in U(0, 1)$
3. $Y = cUg(X)$
4. zwracamy parę (X, Y)

B. Teraz sprawdzamy które pary leżą powyżej krzywej $y = f(x)$ i odrzucamy je. Pozostałe pary leżą pod krzywą (lub na krzywej) więc zgodnie z twierdzeniem mają rozkład równomierny na zbiorze A. Z tego można wywnioskować, że we wszystkich pozostałych nam parach X ma żądaną gęstość f.

Aby metoda była wystarczająco efektywna musi posiadać trzy cechy

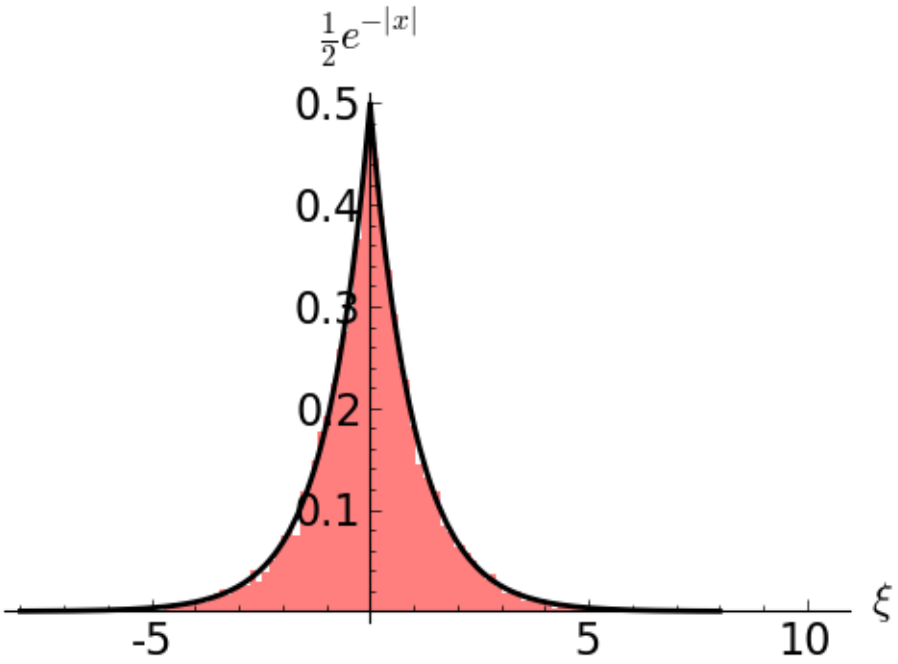
1. Musi istnieć gęstość g i stała c takie, że $f(x) \leq cg(x)$ dla wszystkich $x \in \mathbb{R}$.
2. Rozkład g powinien być stosunkowo łatwy do uzyskania.
3. Prawdopodobieństwo przyjęcia pary powinno być stosunkowo duże (w zasadzie im mniejsze c tym lepiej).

Poniżej podamy kilka pzykładów ilustrujących tą metodę.

Rozkład logistyczny

Spróbujemy wygenerować zmienne losowe o rozkładzie logistycznym bazując na zmiennych o rozkładzie Laplace'a. Najpierw, metodą inwersji generować będziemy liczby losowe o rozkładzie znanym (Laplace'a właśnie),

a następnie przetransformujemy je metodą odrzucania do liczb o końcowym rozkładzie logistycznym bazując na



Rysunek 2.9: Wykres gęstości rozkładu Laplace-a oraz histogram z 10000 prób liczb losowych.

prostym przekształceniu

$$f(\xi) = \frac{1}{2+e^{\xi}+e^{-\xi}} = \frac{1}{2+e^{|\xi|}+e^{-|\xi|}}$$
$$g(\xi) = \frac{1}{2}e^{-|\xi|}$$
$$\frac{g(\xi)}{f(\xi)} = \frac{1}{2}e^{-|\xi|}(1+e^{-|\xi|})^2 \geq \frac{1}{2}$$

Tak więc stawiając $c = 2$ możemy sprawdzać warunek $cUg(\xi)/f(\xi) = U(1+e^{-|\xi|})^2 \leq 1$ co pozwoli nam odrzucać punkty leżące nad krzywą żądanego rozkładu f.

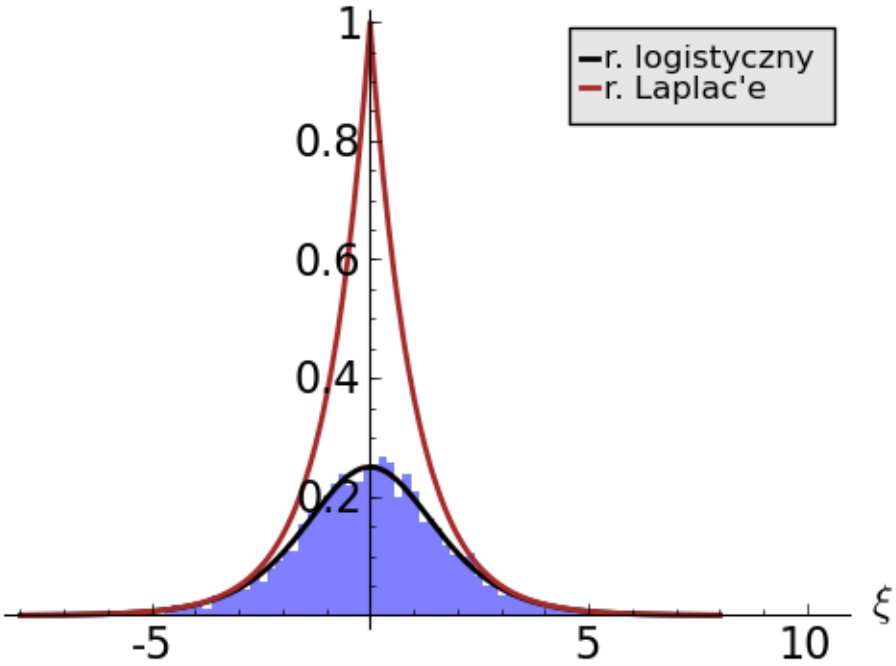
```
# ilosc realizacji
N = 10000
# liczby losowe o rozkladzie Laplace'a (metoda inwersji)
X = [-ln(random()) if random() < 0.5 else ln(random()) for i in xrange(N)]
# liczby losowe o rozkladzie logistycznym (metoda odrzucania)
T = [i for i in X if (1. + e^(-abs(i)))^2*random() <= 1]
```

Zadanie 2.4.1 Zauważając, że $f(x) \leq 1/(4+x^2)$ zbuduj generator liczb losowych o rozkładzie logistycznym. Znajdź poprawne c.

2.3.4 Rozkład normalny

Powyższe metody można z większym lub mniejszym sukcesem stosować do wygenerowania liczb losowych o rozkładzie normalnym (lub jak ktoś woli Gaussa)

$$f(\xi) = \frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$



Rysunek 2.10: Wykres gęstości rozkładu logistycznego oraz histogram z 5078 prób liczb losowych (pozostała reszta z początkowych 10000 została odrzucona).

który parametryzowany jest poprzez swoją średnią μ i wariancję σ^2. Często spotyka się zapis $N(\mu, \sigma)$ oznaczający właśnie powyższy rozkład. My też będziemy posługiwać się takim zapisem, najczęściej oznaczając *N(0,1)* jako rozkład normalny o średniej zero i odchyleniu (choć w tym przypadku i wariancji) równym 1. *N(0,1)* nazywa się standardowym rozkładem normalnym.

Rozkład normalny jest jednym z kilku rozkładów, dla których najłatwiejsza metoda inwersji nie będzie działać wydajnie. Powodem tego jest brak analitycznej postaci funkcji charakterystycznej, która dana jest przykładowo poprzez funkcję błędu

$$F(z) = \frac{1}{2}\left(1 + \operatorname{erf}\ \frac{z}{\sqrt{2}}\right)$$

Zawsze można użyć metody odrzucania (pokażemy jej przykład poniżej) do wygenerowania liczb losowych z *N(0,1)*. Istnieje jednak wiele ciekawych metod rozwiniętych specjalnie dla tego rozkładu. Zaczniemy od mocno akademickiej metody korzystającej z centralnego twierdzenia granicznego.

Centralne twierdzenie graniczne

Zanim podamy sobie owo twierdzenie należy uświadomić sobie fakt, że praktycznie nie korzysta się z tej metody chcąc wygenerować liczby z rozkładem normalnym. Przykład ten jest jednak pouczający, a poza tym z będzie stanowił dobry materiał do odpytywania na egzaminie... Za wikipedią podamy sobie ogólną postać CTG

Twierdzenie Lindeberga-Lévy'ego (CTG) Niech $(X_{n,k})$ będzie schematem serii, w którym $EX_{n,k} = 0$ dla $k \leqslant n$ i dla każdego n mamy $\sum_{k=1}^{n} D^2 X_{n,k} = 1$. Jeśli spełniony jest warunek Lindeberga, tj. dla każdego $\epsilon > 0$ zachodzi

$$\lim_{n\to\infty} \sum_{k=1}^{n} EX_{n,k}^2 \mathbf{1}_{\{|X_{n,k}|>\epsilon\}} = 0,$$

wtedy $\sum_{k=1}^{n} X_{n,k} \xrightarrow{D} N(0,1)$.

Pomimo, że jest ono dosyć skomplikowane (i na dodatek nie podajemy dowodu!), przesłanie jakie niesie wytłumaczyć można w dość prosty sposób:

> Jeżeli $\xi_1, \xi_2, \ldots, \xi_n$ jest ciągiem n niezależnych zmiennych losowych o tych samych rozkładach, to granicznym rozkładem sumy $\xi_1 + \xi_2 + \cdots + \xi_n$ jest rozkład normalny. Podobne stwierdzenia możemy podać dla średnich.

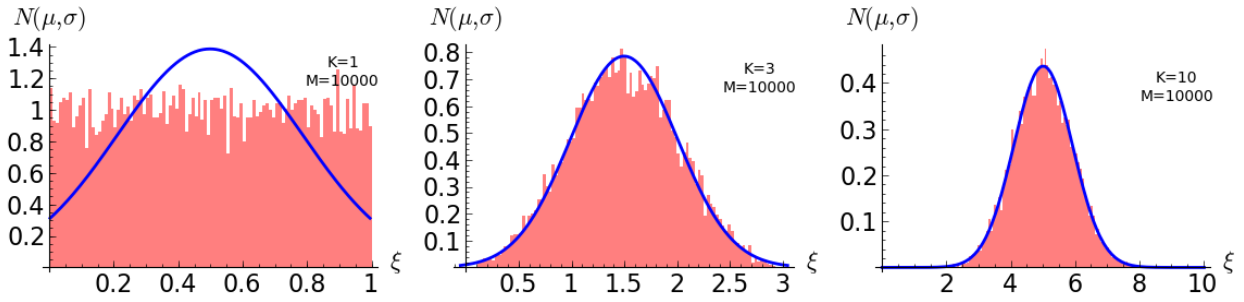
Uwaga 1 CTW nie oznacza, że dla dostatecznie dużej ilości realizacji otrzymamy rozkład normalny. Otrzymamy go tylko dla ich sumy.

Uwaga 2 CTW jest prawdziwe tylko dla zmiennych o rozkładach o skończonej wariancji.

Uwaga 3 CTW jest dobrze spełnione też w nieco ogólniejszym przypadku: dla słabo zależnych od siebie zmiennych oraz dla sumy zmiennych o różnych rozkładach (patrz uwaga 2 powyżej). Jeżeli tylko żadna zmienna losowa nie dominuje w próbie, to suma takich składników będzie miała w przybliżeniu rozkład normalny.

Czas na przykład. Weźmy najprostszy przypadek i popatrzmy na sumy liczb losowych z rozkładem *U(0,1)*.

```
# ilosc zmiennych do sumy
K = 3
# ilosc realizacji poszczegolnej zmiennej
M = 100
# sumy
X = [sum([random() for i in range(K)]) for j in xrange(M)]
```



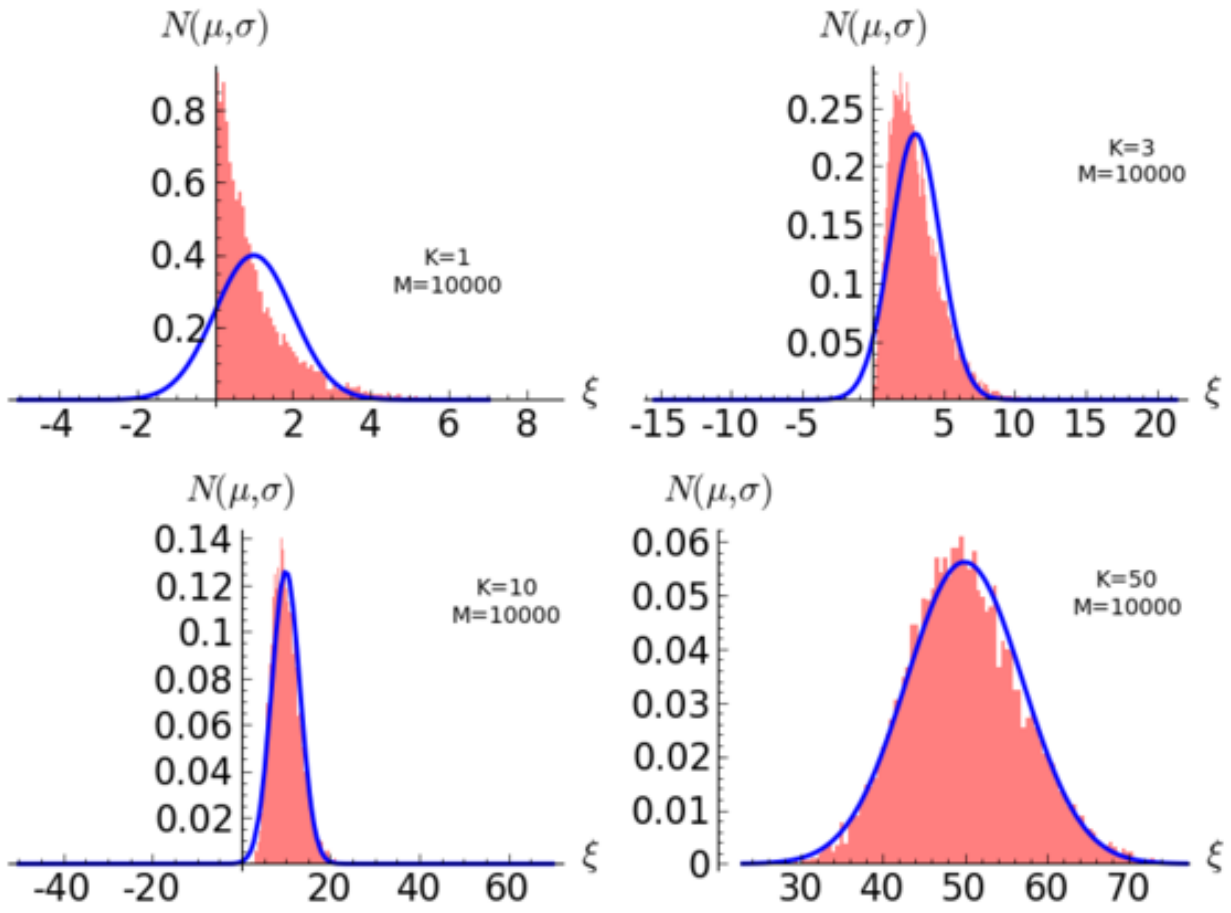
Rysunek 2.11: Wykresy gęstości rozkładu normalnego oraz histogramy z 1000 sum liczb losowych generowanych z *U(0,1)* dla K = 1, 3, 10. Łatwo zauważyć "dostrajanie się" uzyskiwanych histogramów do gęstości rozkładu normalnego w miarę wzrostu liczby składników *K* sumy.

Pomimo dużego podobieństwa histogramów uzyskanych metodą CTG do odpowiednich gęstości spory problem pozostawia generowanie liczb z ogonów (dla dużych wartości argumentów) rozkładu Gaussa. Pomimo, że prawdopodobieństwo uzyskania własnie takich dużych liczb losowych z rozkładem normalnym jest nikłe to jest jednak niezerowe.

Transformacja Boxa-Mullera

Jest to jedna z najstarszych (bo z 1958 roku) i najczęściej stosowanych metod transformacji liczb losowych z *U(0,1)* do *N(0,1)*. W wyniku transformacji z dwóch niezależnych liczb losowych o rozkładzie jednorodnym uzyskiwane są dwie liczby o standardowym rozkładzie Gaussa. Wykorzystuje się tu fakt, że dwuwymiarowy rozkład dwóch niezależnych zmiennych gaussowskich o zerowej średniej jest promieniście symetryczny jeżeli zmienne mają tą samą wariancję. Łatwo to zrozumieć wymnażając przez siebie dwa jednowymiarowe rozkłady

$$e^{-x^2} e^{-y^2} = e^{-(x^2+y^2)} = e^{-r^2}$$



Rysunek 2.12: Wykresy gęstości rozkładu normalnego oraz histogramy z 1000 sum liczb losowych dla K = 1, 3, 10, 50 tym razem pochodzących z rozkładu eksponencjalnego.

Innym sposobem na zrozumienie jak działa algorytm BM to wyobrażenie sobie dwóch uzyskanych zmiennych gaussowskich jako współrzędnych na powierzchni. Wtedy długość wektora to transformacja pierwszej liczby losowej $U_1 \in U(0,1)$ a jego faza to proste wymnożenie drugiej zmiennej $U_2 \in U(0,1)$ przez 2π.

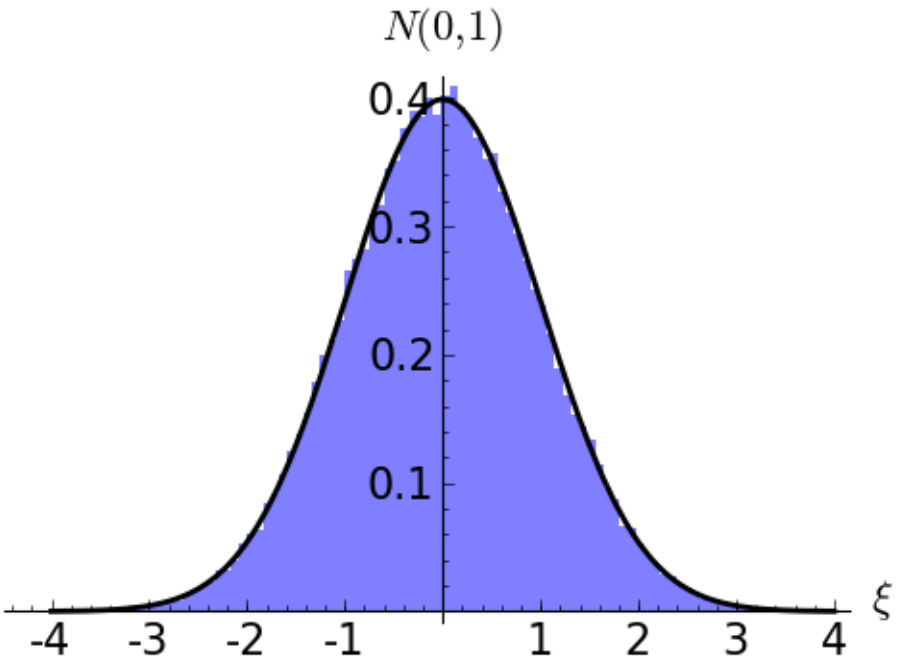
Algorytm

1. Wylosuj dwie liczby $(U_1, U_2) \in U(0,1)$.
2. Zwróć $(n_1, n_2) = (\sqrt{-2\ln(U_1)}\sin(2\pi U_2), \sqrt{-2\ln(U_1)}\cos(2\pi U_2)$

Jako, że algorytm produkuje po jednym obiegu dwie liczby losowe, procedury realizujące go najczęściej zwracają pierwszą liczbę losową a drugą przytrzymują w pamięci by zwrócić ją w kolejnym wywołaniu procedury. Realizacja w Sage prostej wersji (bez trzymania drugiej liczby w cache-u) nie powinna powodować większych trudności. Dla odmiany napiszmy sobie prostą funkcję `box_muller` zwracającą *k* liczb o standardowym rozkładzie normalnym.

```
def box_muller(k=1):
  ret = []
  for i in xrange(0,k,2):
    a, b = sqrt(-ln(random()) * 2), 2 * pi.n() * random()
    ret.append(a*sin(b))
    ret.append(a*cos(b))
  return ret[:-1] if k%2 else ret
```

Zobaczmy, czy rzeczywiście histogram z *10000* liczb pokryje się z *N(0,1)*.



Rysunek 2.13: Wykresy gęstości rozkładu normalnego oraz histogram z 10000 liczb losowych przetransformowanych z *U(0,1)* do *N(0,1)* z wykorzystaniem transformacji Boxa-Mullera.

Aby wyjasnić dlaczego taka oto transformacja ma prawo bytu można (za Boxem i Mullerem) skorzystać najpierw z transformacji odwrtotnej zmiennych n_1, n_2

$$U_1 = \exp\left(-\frac{n_1^2 + n_2^2}{2}\right)$$
$$U_2 = -\frac{1}{2}\arctan\frac{n_2}{n_1}$$

I następnie policzyć rozkład dwuwymiarowy zmiennych n_1, n_2 korzystając z Jakobianu przekształcenia

$$f(n_1, n_2) = U(0,1) * |J|,$$
$$J = \Big| \frac{\partial(U_1, U_2)}{\partial(n_1, n_2)} \Big|,$$

z czago łatwo policzyć rozkład dwuwymiarowy

$$f(n_1, n_2) = \frac{1}{2\pi} \exp\Big(-\frac{(n_1^2 + n_2^2)}{2}\Big) = \frac{1}{\sqrt{2\pi}} \exp\Big(-\frac{n_1^2}{2}\Big) \frac{1}{\sqrt{2\pi}} \exp\Big(-\frac{n_1^2}{2}\Big) = f(n_1) f(n_2),$$

i wywnioskować niezależność zmiennych n_1, n_2.

W miarę rozrastania się podręcznika w tym miejscu powinny pojawić się inne, znacznie ciekawsze metody uzyskiwania zmiennych (czy może raczej liczb) losowych, na przykład: przybliżanie kawałkami liniowe z użyciem rozkładów trójkątnych, metoda Mony-Pythona, szereg metod odrzucania (biegunów, Marsaglia-Bray, GRAND) czy metoda rekursywna Wallece-a. Innym zagadnieniem jest problem wytwarzania liczb losowych dla gaussowskich ogonów. Na duże (w sensie modulo, czyli gdy nie patrzymy na ich znak) liczby gaussowskie mające bardzo małe prawdopodobieństwo wystąpienia mówimy, że leżą w ogonach rozkładu Gaussa. Do takich przypadków również powstały interesujace algorytmy. Pominiemy natomiast testy i porównywanie generatorów, jako że nie o tym jest ten wykład (choć zagadnienie jest na pewno ciekawe).

2.4 Zadania

Zadanie 2.Z.1 Wykorzystaj metodę Boxa-Mullera do sprawdzenia jaki procent liczb gaussowskich o rozkładzie $N(\mu, \sigma)$ wypada poza $\pm\sigma, \pm 3\sigma, \pm 6\sigma$.

Zadanie 2.Z.2 TBA

Symulacje procesów losowych dyskretnych

Relizacje komputerowe dowolnych procesów rynkowych zawsze będą dyskretne. Co prawda opisujące owe procesy modele będą zarówno dyskretne jak i ciągłe, ale końcowe algorytmy zawsze będą bazować na dyskretnych ich realizacjach. Zaczniemy od prostego, mało realistycznego modelu, w którym cena produktu, waloru będzie mogła rosnąć lub maleć w każdym kroku czasowym realizacji.

3.1 Prosty model dynamiki ceny

ARMA

Analizując dane pochodzące z szeroko pojętych źródeł rynków finansowych dość często korzysta się z Modeli Autoregresyjnych ze Średnią Kroczącą. Angielska nazwa takich modeli to Autoregresive Moving Average Models i stąd nazwa ARMA. Zwykle procedura polega na pewnej analizie posiadanych danych i dopasowaniu do nich parametrów takiego modelu. Takim zagadnieniem zajmuje się Analiza Szeregów Czasowych. Szeregów dostarcza "samo życie", czyli np: giełda. Jako, że ten kurs ma na celu modelowanie rynków, czyli ma *generować* takie szeregi, pobawimy się teraz w symulowanie rynku.

Model ARMA to nic innego jak układ równań różnicowych ze stałymi współczynnikami.

Stochastyczne równania różniczkowe

Stochastyczne równania różniczkowe (SDE, od angielskiej nazwy Stochastic differential equations) są obecnie uważane za standardowe narzędzie wykorzystywane do analizy niektórych wielkości opisujących dynamikę rynków finansowych. Do tych wielkości należą ceny aktywów, stopy procentowe czy ich pochodne. W przeciwieństwie do zwyczajnych równań różniczkowych, które posiadają jendoznaczne rozwiązanie, rozwiązaniami SDE są ciągłe w czasie procesy stochastyczne. Metody komputerowe wykorzystywane do analizy SDE bazują na klasycznych metodach wykorzystywanych do rozwiązywania tradycyjnych, deterministycznych równań różniczkowych, są jednak uogólnione tak, aby radzić sobie z procesami losowymi.

Zestaw zmiennych losowych X_t indeksowanych liczbami rzeczywistymi *t* nazywamy procesem losowym ciągłym (ze względu na czas). Każda *realizacja* procesu losowego to przypadkowa wartość zmiennej losowej X_t dla każdego *t*, jest więc funkcją czasu. Co ciekawe, *każda* deterministyczna funkcja $f(t)$ może być uważana za proces stochastyczny, którego wariancja znika.

Najbardziej znanym przykładem procesu losowego szeroko występującego w modelach fizyki, chemii ale i rynków finansowych jest *proces Wienera* $W(t) = W_t$, ciągły proces stochastyczny posiadający nastepujące własności

1. jest to proces rzeczywisty,
2. startuje z zera ($W_0 = 0$),
3. ma stacjonarne i niezależne przyrosty na nieprzekrywających się przedziałach,
4. jest procesem Gaussa o zerowej wartości średniej $\langle W_t - W_s \rangle = 0$ i wariancji przyrostów $\langle [W_t - W_s]^2 \rangle = 2D(t-s)$,
5. proces Wienera może być reprezentowany ciągłymi trajektoriami.

Wynika z tego, że dla każdej różnicy czasów *t - s* zmienna losowa $W_t - W_s$ jest zmienną losową gaussowską o zerowej wartości średniej i wariancji $2D(t-s)$. Więc jego rozkład prawdopodobieństwa ma postać

$$f_{W_t - W_s}(x) = \frac{1}{\sqrt{2\pi D(t-s)}} \exp\Big[- \frac{x^2}{4D(t-s)} \Big].$$

Proces taki może być wyprowadzony jako proces graniczny błądzenia przypadkowego. Wystarczy tylko zbadać granicę dla której wielkość skoku i czas pomiędzy skokami będą maleć do zera. Tak zdefiniowanym procesem posługujemy się zwyczajowo, gdy podczas analizy probalmu pojawia się jakaś nieregularna siła czy zaburzenie którego nie możemy opisać równaniami deterministycznymi.

Typowe dla rynków finansowych *równanie dyfuzji* może być modelowane przez równanie różniczkowe posiadające część deterministyczną zwaną **dryftem** oraz część losową zwaną **dyfuzją**. Ta ostatnia jest bardzo często reprezento-

wana właśnie przez proces Wienera. Możemy sobie napisać ogólne równanie

$$dX = a(t, X)dt + b(t, X)dW_t.$$

Jest to postać różniczkowa. W zwykłych równaniach różniczkowych zazwyczaj stosujemu pochodne dx/dt. W tym przypadku postać różniczkowa ma większy sens, jako, że wiele interesujących nas procesów losowych (jak ruch Browna) są procesami ciągłymi aczkolwiek nie są różniczkowalne. Powyższe równanie nabiera większego sensu pod znakiem całki

$$X(t) = X(0) + \int_0^t a(s, y)ds \int_0^t b(s, y)dW_s.$$

Ostatni wyraz z prawej zwany jest całką Ito.

5.1 Równanie Blacka-Scholesa

Jednym z bardziej znanych, historycznym już równaniem stochastycznym, jest równanie opisujące geometryczny ruch Browna

$$dX = \mu X dt + \sigma X dW_t,$$

gdzie μ, σ to wielkości stałe. Równanie to jest jednym z podstawowych elementów modelu wyceny opcji Blacka-Scholesa. Teoria ta została nagrodzona Nagrodą Nobla z ekonomii w roku 1997, a opracowana przez absolwenta fizyki i doktora matematyki Fischera Blacka oraz ekonomistę Myrona Scholesa. Teoria Blacka-Scholesa pozwala na wycenę wartości tzw. finansowych instrumentów pochodnych, czyli opcji, oraz służy do optymalizacji "bezpiecznego" portfela inwestycyjnego.

Pomimo tego, że równanie to wygląda na proste, żeby nie powiedzieć zbyt proste na to by opisywać jakąkolwiek rzeczywistość na rynkach finansowych, ma ono olbrzymie znaczenie, jako, że może być rozwiązane ściśle dając wynikowy wzór z którego możemy wyliczyć zmianę cen prostych opcji. Jak już powiedzieliśmy, rozwiązanie jest geometrycznym ruchem Browna

$$X(t) = X_0 \exp\Big[\Big(\mu - \frac{\sigma^2}{2}\Big)t + \sigma dW_t\Big].$$

Dzięki zamkniętej postaci rozwiązania mamy możliwość testowania metod numerycznych, które przedstawimy poniżej.

5.2 Schemat Eulera-Maruyamy

Najprostszą metodą numerycznego rozwiązywania równań różniczkowych zwyczajnych jest metoda Eulera. Bazuje ona np. na rozwinięciu Tylora w pierwszym rzędzie przybliżenia. Stochastycznym analogiem tej metody jest metoda Eulera-Maruyamy.

Będziemy chcieli podać przybliżone rozwiązanie ogólnej postaci SDE na przedziale czasowym $t \in [t_0, t_E]$. Na początku zdyskretyzujemy sobie ów przedział czasowy, ustalając na siatkę *N* punktów

$$t_0 < t_1 < t_2 < \cdots < t_{N-2} < t_E.$$

Dążymy do tego, aby na tej siatce znaleźć przybliżone wartości zmiennej X. Oznaczmy je

$$w_0 < w_1 < w_2 < \cdots < w_{N-2} < w_E.$$

Są to oczywiście przybliżone rozwiązania zmiennej x dla odpowiednich czasów z powyższej siatki $\{t_i\}$. Zakładając wartość początkową dla SDE $X(t_0) = X_0$ możemy pokusić się o rozwiązanie numeryczne w następującej postaci

$$w_0 = X_0$$
$$w_{i+1} = w_i + a(t_i, w_i)\Delta t_{i+1} + b(t_i, w_i)\Delta W_{i+1}$$
$$\Delta t_{i+1} = t_{i+1} - t_i$$
$$\Delta W_{i+1} = W(t_{i+1}) - W(t_i).$$

Kluczową sprawą w tym punkcie jest problem: jak zamodelować ΔW_i? Mając do dyspozycji generator liczb losowych z rozkładem *N(0,1)* każdą losową liczbę ΔW_i obliczamy ze wzoru

$$\Delta W_i = \sqrt{\Delta t_i} z_i,$$

gdzie z_i jest losowana właśnie z *N(0,1)*. Aby scałkować proces stochastyczny użyjemy formuły na przyrost procesu Wienera

$$\int_{t_i}^{t_{i+1}} \Gamma(t)dt = \int_{t_i}^{t_{i+1}} dW(t) = W(t_{i+1}) - W(t_i)$$

Z definicji procesu Wienera wiemy, że jest on procesem Gaussa o zerowej średniej i wariancji liniowej w czasie $\langle [W(t_{i+1}) - W(t_i)]^2 \rangle = 2D\Delta t_i$, co daje nam w sensie średnio-kwadratowym $\Delta W \propto \sqrt{\langle [\Delta W(t)]^2 \rangle} = \sqrt{\Delta t_i}$. Scałkowanie procesu Wienera prowadzi do

$$\int_{t_i}^{t_{i+1}} dW_t = \sqrt{\Delta t_i} z_i.$$

Jeżeli założymy sobie, że krok czasowy (odległości na siatce rozwiązań) jest stały i wynosi $\Delta t_i = h$ możemy napisać schemat explicite

$$w_{i+1} = w_i + ha(t_i, w_i) + \sqrt{h} b(t_i, w_i) z_i.$$

Jako, że każdy zestaw wartości $\{w_i\}, i = 0, \ldots, E$ wyprodukowany przez powyższą formułę będzie przybliżonym rozwiązaniem procesu losowego, to i każda realizacja (każdy zestaw) będzie również losowa, a co za tym idzie - każda realizacja procesu będzie inna.

5.3 Schemat Millsteina

Dodaje on poprawkę do poprzedniego rozwiązania, powodując, że schemat staje się schematem pierwszego rzędu w sensie silnym. Dany jest on wzorem iteracyjnym

$$w_0 = X_0$$
$$w_{i+1} = w_i + a(w_i, t_i)h - \frac{h}{2} b(x_i, t_i) \frac{\partial b'}{\partial x}(w_i, t_i)(z_i^2 - 1) + \sqrt{h} b(w_i, t_i) z_i.$$

Obie metody (Millsteina i Eulera-Maruyamy) redukują się do tego samego schematu gdy część losowa nie jest zależna od zmiennej x. Jeżeli zależność istnieje, schemat Millsteina będzie szybciej zbieżny od schematu EM.

Przykłady całkowania procesów stochastycznych

6.1 Proces dyfuzji

Jest to prawdopodobnie najprostszy proces stochastyczny wykorzystujący biały szum gaussowski jako proces losowy. Przez matematyków nazywany jest po prostu procesem Wienera ponieważ prawa strona równania ruchu zawiera tylko i wyłącznie ów proces. Z drugiej strony jest obok procesu Poissona najważniejszym procesem losowym na bazie którego można zdefiniować całą rodzinę procesów losowych o ciągłych realizacjach. Równanie to można przedstawić używając równania Ito

$$dx(t) = \sqrt{2D}dW(t).$$

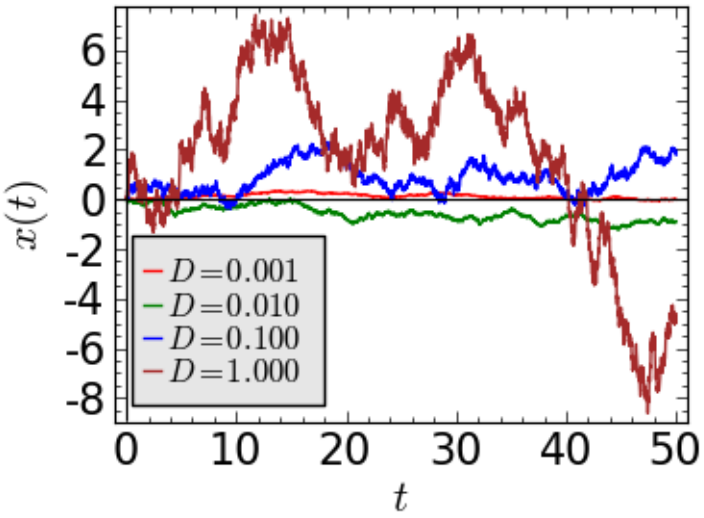
Realizacja jest funkcją ciągłą, ale nigdzie nieróżniczkowalną (jako że pochodna procesu Wienera nie istnieje). Za pomocą znanego już schematu Eulera-Maruyamy (EM) możemy sobie wygenerować pojedynczą realizację takiego procesu. Parametr D reguluje natężenie szumu.

$$x_0 = 0$$
$$x_{i+1} = x_i + \sqrt{2hD}N(0,1).$$

Wiemy, że dla procesu Wienera $W(0) = 0$, wystartujemy więc z $x(0) = 0$. Weźmy krok $h = 0.01$ i 5000 kroków czasowych. Dla przejrzystości weźmiemy natężenie szumu $D = 1$. Jako, że wiemy jak generować zmienne z rozkładem *N(0,1)* użyjemy sobie "symbolicznego" oznaczenia na funkcję zwracajacą takie zmienne. Funkcję taką nazwiemy `std_norm`. Konkretna realizacja takiej funkcji może odbywać np: poprzez algorytm Boxa-Mullera. Funkcja ta będzie przy wywołaniu zwracała jedną liczbę losową z *N(0,1)*.

```
h = 0.01
N = 5000
x0 = 0
D = 1

x = [x0]
for i in xrange(1,N):
  n01 = std_norm()
  x.append(x[i-1] + sqrt(2*h*D) * n01)
```

Teraz narysujmy sobie takie realizacje dla kilku różnych wartości parametru *D*.



Rysunek 6.1: Proces dyfuzji dla kilku różnych wartości parametru D.

Na pierwszy rzut oka trajektorie (czy realizacje, przebiegi...) wyglądają kompletnie inaczej. Dla małych wartości D krzywe są bardziej regularne niż dla tych parametryzowanych przez większe wartości D, dla których to wykres jest mocno poszarpany i nieregularny. Jeżeli jednak narysowalibyśmy je osobno, nie oznaczając osi, identyfikacja byłaby niemożliwa - nie widzimy bowiem relacji pomiędzy wartościami (przyrostami).

Jako, że rozwiązanie równania dyfuzji znane jest od dawoen dawna, możemy potestować na ile dokładnie rozwiązujemy nasze równanie. Oczywiście jedyne co możemy określić, to rozkład gęstości prawdopodobieństwa. Zakładając warunki początkowe

$$P(x,0) = \delta(x)$$

co oznacza, że cały zespół cząstek podlegających dyfuzji wystartuje z $x(0) = 0$, możemy podać odpowiedź, bazując na równaniu dyfuzji

$$\frac{\partial P(x,t)}{\partial t} = D\frac{\partial^2 P(x,t)}{\partial x^2}.$$

Wynikiem jest rozkład Gaussa postaci

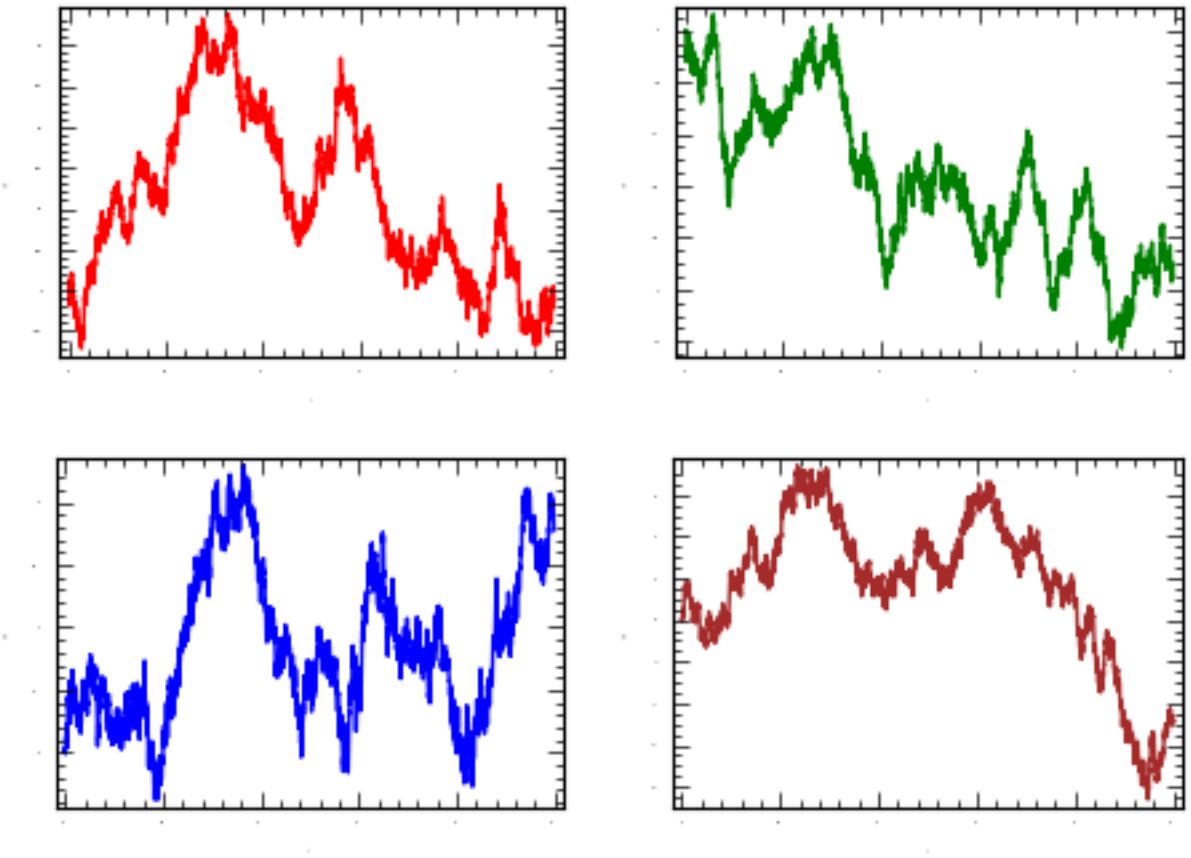
$$P(x,t) = \frac{1}{\sqrt{4\pi Dt}} \exp\Big[-\frac{x^2}{4Dt}\Big].$$

w którym wariancja rośnie liniowo z czasem a średnia jest równa zero. Zobaczmy, czy jesteśmy w stanie zweryfikować powyższy wzór numerycznie. Postaramy się znaleźć histogram pozycji 10000 cząstek po 100 krokach i porównamy go z powyższym wynikiem. Założymy sobie $D = 0.1$ i krok czasowy $h = 0.01$. Oznacza to, że po 100 krokach symulacji osiągniemy rzeczywisty czas równy 1.

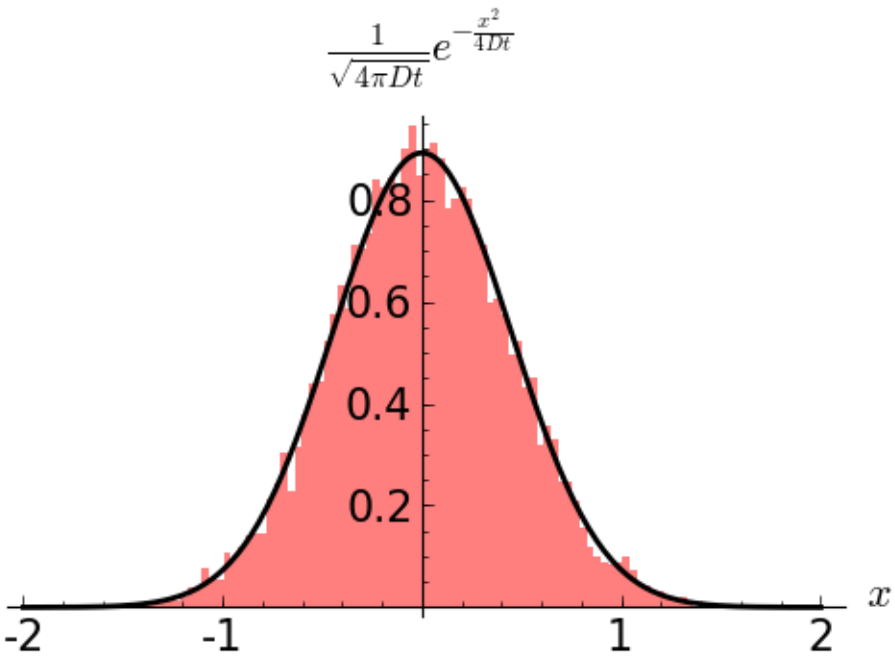
Zadanie 6.1.1 Wykonaj w Sage powyższy rysunek. Metodę generowania histogramów znajdziesz w pierwszej części skryptu. Listę położeń, którą należy podać do histogramu wygeneruj w podobny sposób jak na początku tego rozdziału.

6.2 Dyfuzja ze stałym dryftem

Proces taki otrzymuje się bezpośrednio jako graniczny przypadek niesymetrycznego błądzenia przypadkowego (polecam poczytać ten podrozdział). Równanie opisujące ten proces ma postać zbliżoną do poprzedniego, bogatsze jest



Rysunek 6.2: Proces dyfuzji dla kilku różnych wartości parametru D. Są to te same przebiegi co w poprzednim wykresie. Kolory tu uzyte odpowiadają kolorom z wykresu poprzedniego.



Rysunek 6.3: Proces dyfuzji po 100 krokach symulacji dla $D = 0.1$, $h = 0.01$, $N = 100$ a co za tym idzie $t = Nh$.

jednak o dodatkowy czynnik, zwany dryftem

$$\frac{\partial P(x,t)}{\partial t} = -V\frac{\partial P(x,t)}{\partial x} + D\frac{\partial^2 P(x,t)}{\partial x^2}.$$

Rozwiązaniem równania dyfuzji z dryfem, z takim samym warunkiem początkowym jak poprzednio (wszystkie cząstki, bądź realizacje zaczynają z tego samego położenia $x(0) = 0$ jest następujaca funkcja

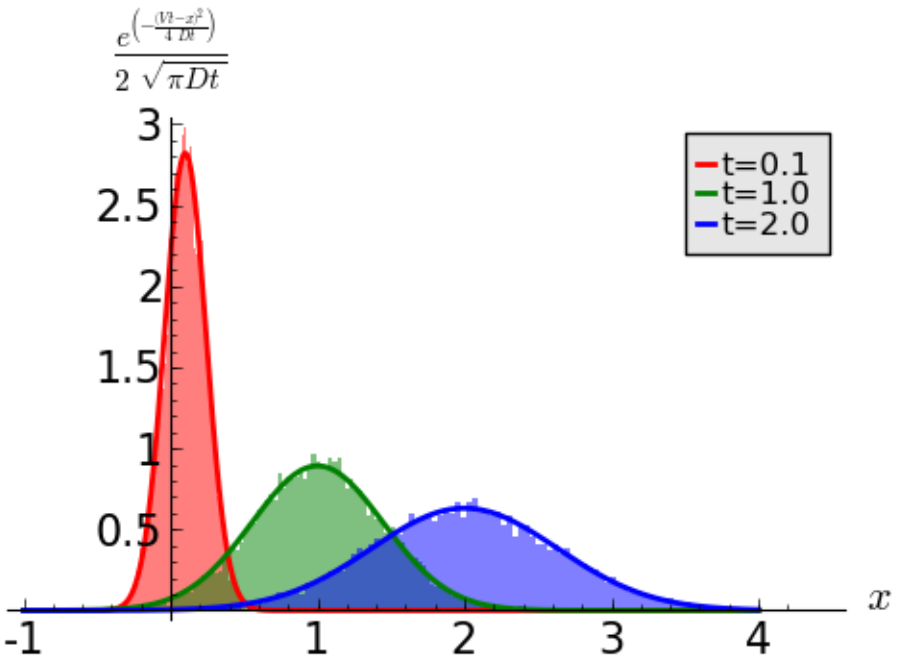
$$P(x,t) = \frac{1}{\sqrt{4\pi Dt}} \exp\Big[-\frac{(x-Vt)^2}{4Dt}\Big].$$

Jest to funkcja Gaussa opisujaca zmienne losowe normalne, a dwa pierwsze momenty wynoszą odpowiednio $\xi(t) = Vt$ oraz $\sigma_\xi^2 = 2Dt$. Łatwo zauważyć, że zarówno średnia jak i wariancja zależne są liniowo od czasu. Ponadto wariancja jest identyczna jak w procesie dyfuzji bez dryftu. Dryft ów stały można z punktu widzenia fizyki rozumieć jako stałą siłę przyłożoną do cząstki (coś na kształt cząstki umieszczonej na równi pochyłej) - położenie cząstki rośnie liniowo z czasem (jak w ruchu jednostajnie prostoliniowym), ale fluktuacje rosną w czasie jak pierwiastek $\sqrt{t}$.

Podobną analizę numeryczną jak poprzednio możemy przeprowadzić i tutaj. Tym razem, wykreślimy sobie stroboskopowo histogram położeń po kilku krokach: *N = 10, 100, 200*. Po lekkiej modyfikacji numeryczny schemat EM będzie wyglądał tak

```
h = 0.01
N = 5000
x0 = 0
V = 1
D = 1

x = [x0]
for i in xrange(1,N):
  n01 = std_norm()
  x.append(x[i-1] + V*h + sqrt(2*h*D) * n01)
```

Teraz wystarczy zobaczyć, czy histogramy położeń po czasie *t=0.1, 1, 2* będą odpowiadały obliczonej powyżej funkcji rozkładu.



Rysunek 6.4: Proces dyfuzji ze stałym dryftem po 10, 100 i 200 krokach symulacji dla $D = 0.1$, $h = 0.01$, a co za tym idzie $t = 0.1, 1, 2$.

Możemy policzyć sobie teraz średnie, odchylenie standardowe oraz błędy względny i bezwzględny przybliżeń dokładnych rozwiązań procesu dyfuzji z dryftem.

czas		**teoria**	**symulacje**	E_b	E_w **[%]**
t=0.1	średnia	0.10	0.09972	0.0002766	0.2766
	std	0.1414	0.1413	0.00007544	0.05335
t=1	średnia	1.0	1.005	0.005345	0.5345
	std	0.4472	0.444	0.003026	0.6766
t=2	średnia	2.0	2.001	0.001460	0.07302
	std	0.6324	0.6362	0.003708	0.5863

Jak widzimy błędy bezwzględne dochodzą do około pół punktu procentowego różnicy dla 10000 realizacji. Zwiekszenie próby spowoduje jeszcze lepsze dopasowanie, zmniejszenie spowoduje większe odchylenia od wartości rzeczywistych.

Zadanie 6.1.2 Oblicz błędy przybliżenia rozwiazania problemu dyfuzji ze stałym dryftem dla 10, 100, 500 i 1000 różnych realizacji. Zestawienia podaj w tabeli.

6.3 Proces Ornsteina-Uhlenbecka

6.4 Równanie Blacka-Scholesa

Dodatek matematyczny

7.1 Całka Ito

TBA

Rozwiązywanie numeryczne równań różniczkowych zwyczajnych

8.1 Metoda Eulera

Jest to bodaj najprostszy sposób na numeryczne rozwiązywanie równań różniczkowych zwyczajnych. Technicznie to metoda pierwszego rzędu. Bazuje na prostej interpretacji definicji pochodnej. Rozpatrujemy równanie postaci

$$\frac{dy}{ds} = y' = f(y, s),$$

z zadanymi warunkami początkowymi $(sx(0), y(0)) = (x_0, y_0)$. Stosując przekształcenie

$$\frac{dy}{ds} = \lim_{\Delta s \to 0} \frac{\Delta y}{\Delta s}$$

Na chwilę zapomnijmy o tej granicy. Mamy

$$\frac{\Delta y}{\Delta s} = \frac{y(s + \Delta s) - y(s)}{\Delta s}$$

stawiając krok "czasowy" $\Delta x = h$ (jak to zwyczajowo w symulacjach komputerowych) dostajemy

$$\frac{y(s + h) - y(s)}{h} \simeq f(y, s),$$
$$y(s + h) \simeq y(s) + f(y, s)h.$$

Oczywiście pełna równość zachodzi tylko w granicy $h \to 0$. Na potrzeby numeryczne jednak nie musimy się tym przejmować. Należy mieć tylko świadomość, że zmniejszanie kroku czasowego zbliża nas do wyniku dokładnego (zazwyczaj). Jako, że w iteracjach "czasowych" wykorzystujemy stały krok h, więc w danej, powiedzmy *i*-tej iteracji czas rzeczywisty zastąpiony zostanie poprzez $s_i = i \cdot h$, możemy równie dobrze wprowadzić indeksowanie nie po czasie *s* ale po zmiennej iteracyjnej *i*

$$y(s = ih) = y(s_i) = y_i, \quad s_i = ih,$$
$$y(ih + h) = y(h(i + 1)) = y(ih) + f(y(ih), ih)h,$$
$$y_{i+1} = y_i + f(y_i, s_i)h.$$

Dostając tzw. schemat Eulera. Innymi metodami wyprowadzenia tego prostego schematu będzie użycie rozwinięcia Tylora lub wycałkowanie równania różniczowego od s_0 do :math‘s_0 + h‘ za pomocą metody prostokątów (uzywając tylko jednego prostokąta na całym przedziale całkowania).

Zadanie D1.1 Uzyskaj schemat Eulera z rozwinięcia Tylora.

Zadanie D1.2 Uzyskaj schemat Eulera całkując równanie różniczkowe.

Dokładność metody Eulera mocno zależy od wyboru kroku całkowania h. Musimy też na początku zadać warunki startowe (początkowe) iteracji, zupełnie jak podczas rozwiązywania równań metodami analitycznymi. Dla przykładu obliczmy numerycznie ruch przetłumionego sprężyny o współczynniku sprężystości k.

$$\dot{x}(t) = -kx(t).$$

Rozwiązaniem jest zanik eksponencjalny $x(t) = x_0 \exp(-kt)$. Wybierzemy sobie warunek początkowy $x_0 = 1$ i współczynnik sprężystości $k = 0.1$. Policzmy N iteracji i narysujmy wykres. Na początku napiszemy dyskretną wersję równania ruchu

$$x_{i+1} = x_i - kx_i h = x_i(1 - kh).$$

Teraz kod Sage

```
# liczba iteracji rownania
N = 300
# czas poczatkowy
t = 0
# skok czasowy
h = 0.01
# parametr rownania (sprezystosc)
k = 1
# wartosc poczatkowa x(t=0)
x_0 = 1
#inicializacja listy poprzez wartosc poczatkowa
lista_x = [x_0]
for i in xrange(N):
  lista_x.append(lista_x[i] * (1 - k*h))
```

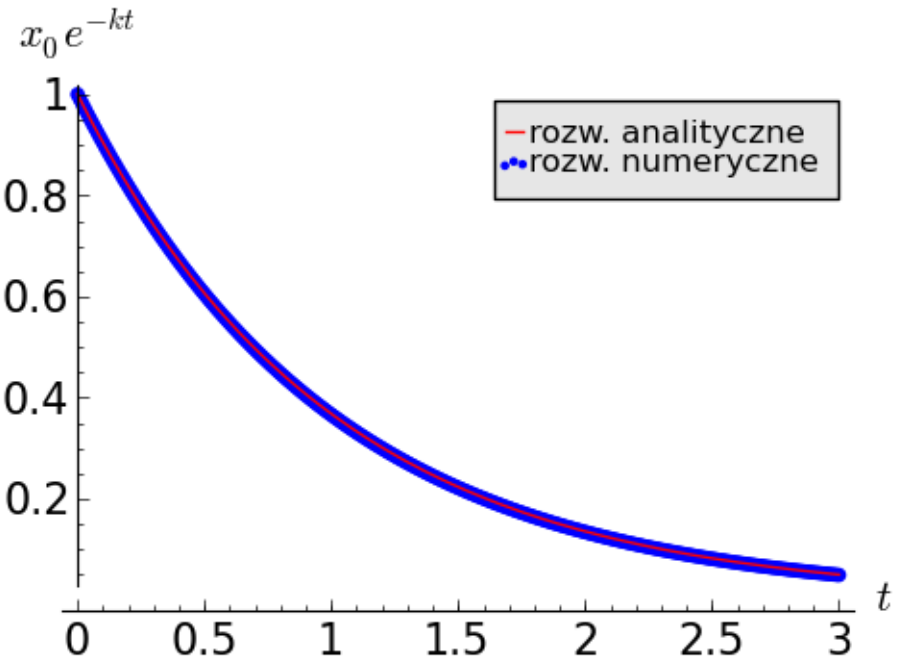
wynik (rozwiąznanie numeryczne równania) trzymany jest w liście `lista_x`. Wyrysujmy ją standardowo, razem z wynikiem analitycznym.

Pomimo użycia prostej metody (pierwszego rzędu), wykresy wyglądają identycznie. No, ale czy na pewno jest tak pięknie? Poprawność metody możemy łatwo zbadać obliczając błędy względny (E_w) i bezwzględny (E_b).

$$E_b = \bar{y} - y,$$
$$E_w = \frac{\bar{y} - y}{\bar{y}} = \frac{E_b}{\bar{y}}.$$

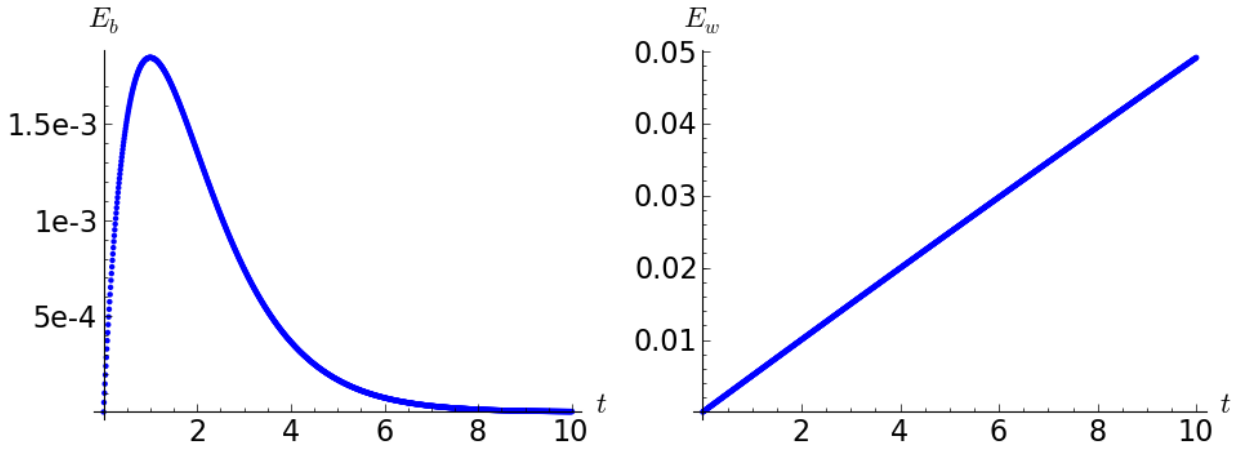
gdzie y to wielkość obliczona algorytmem a $\bar{y}$ to dokładna wielkość analityczna. Dla jasności - nie interesują nas w tym przypadku znaki błędów a jedynie ich wartość bezwzględna (tu proszę zwrócić uwage na nomenklaturę, żeby nie pomylić wartości bezwzględnej z błędem bezwzględnym). Dlatego najczęściej oblicza się nie E_w a $|E_w|$. Wymaga to drobnej korekty powyższego kodu

```
N, t, h, k, x_0 = 1000, 0, 0.01, 1, 1
g(s) = x_0*exp(-k*s)
lista_x = [x_0]
Eb = [g(0) - x_0]
Ew = [Ew[0]/g(0)]
for i in xrange(1,N):
  lista_x.append(lista_x[i-1] * (1 - k*h))
  Eb.append(abs(g(i*h) - lista_x[i]))
  Ew.append(Eb[i]/g(i*h))
list_plot(zip([i*h for i in xrange(N+1)],Eb)).show()
list_plot(zip([i*h for i in xrange(N+1)],Ew)).show()
```



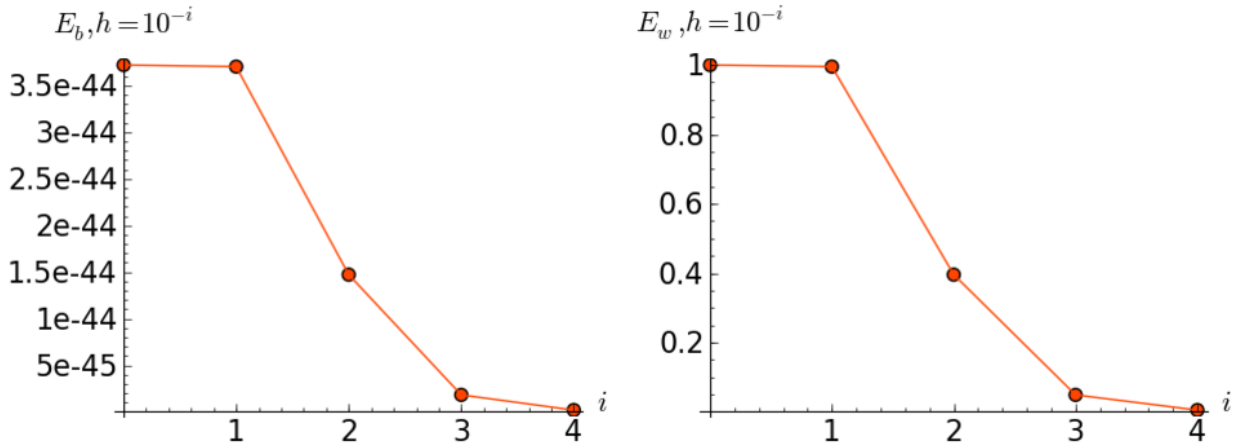
Rysunek 8.1: Rozwiązanie analityczne i numeryczne równania $\dot{x} = -kx$.

Spójrzmy. Na pierwszym wykresie odłożony mamy błąd bezwzględny. Widzimy, że dla krótkich czasów odbiega on od wartości analitycznej by dla większych czasów zmaleć do zera. Mogą być tego 2 powody: (i) różnica pomiędzy obiema wartościami maleje zo zera lub (ii) obie wartości maleją do zera, więc ich różnica też. Jako, że funkcja jest eksponencjalna, dużo bardziej prawdopodobny jest ten drugi scenariusz. Aby zobaczyć, czy błąd rośnie z ilością iteracji (w czasie) wykreślimy błąd względny. Mówi on nam o stosunku błedu bezwzględnego do wartości analitycznej (rysunek po prawej). Tu jak widać rośnie on wraz z czasem, z czego możemy wywnioskować, że wraz z ilością iteracji coraz mniej dokładnie obliczamy wartość y.



Rysunek 8.2: Błąd bezwzględny (po lewej) i błąd względny (po prawej).

Najprosztszą metodą poprawienia jakości rozwiązań jest zmniejszenie kroku całkowania. Zależności pozostaną podobne, zmniejszy się jednak wartośc błędów w danej chwli czasowej.



Rysunek 8.3: Błąd bezwzględny (po lewej) i błąd względny (po prawej) po czasie $t = 10$ dla różnych kroków czasowych $h = 1, 0.1, 0.01, 0.001, 0.0001$.

W tabeli zawarto wartości błędów bezwzględnego i względnego dla różnych wielkości kroku czasowego symulacji, po osiągnięciu czasu końcowego $T_E = 10$. Widać, że pomimo, że za każdym zmniejszeniem kroku zwiększała się ilość kroków czasowych, dokładność obliczeń rosła - malał zarówno błąd bezwzględny jak i względny.

h	N	E_b	E_w
1	100	3.7210^{-44}	1
0.1	1 000	3.7010^{-44}	0.995
0.01	10 000	1.4710^{-44}	0.396
0.001	100 000	1.8210^{-45}	0.0488
0.0001	1 000 000	1.8610^{-46}	0.0499

Przejdźmy teraz do rozwiązania równania różniczkowego wyższego stopnia. Znów posłużymy się przykładem oscylatora harmonicznego. Tym razem rozwiążemy równanie Newtona dla punktowej cząstki o masie m w potencjale $U(x) = -kx^2/2$. Pominiemy siły tarcia. Również w spokoju zostawimy wymuszenie.

$$m\ddot{x}(t) = -kx(t).$$

Równanie to posiada znane analityczne rozwiązanie. Oznaczając $\omega_0^2 = k/m$ dostajemy

$$x(t) = A\sin(\omega_0 t) + B\cos(\omega_0 t).$$

Stałe *A, B* (amplitudy) zależne są od wyboru warunków początkowych. Spróbujmy numerycznie rozwiązać równanie ruchu tak, aby pokazać zgodność z rozwiązaniem. Aby napisać schemat Eulera dla równania drugiego stopnia najpierw trzeba przepisać równanie do układu równań na x i $v = \dot{x}$.

$$\dot{x}(t) = v(t),$$
$$\dot{v}(t) = -\frac{k}{m}x(t) = -\omega_0^2 x(t).$$

Teraz wystarczy zdyskretyzować te równania, tak samo jak robiliśmy to z równaniem pierwszego rzędu.

$$x_{i+1} = x_i + v_i h,$$
$$v_{i+1} = v_i - \omega_0^2 x_i h.$$

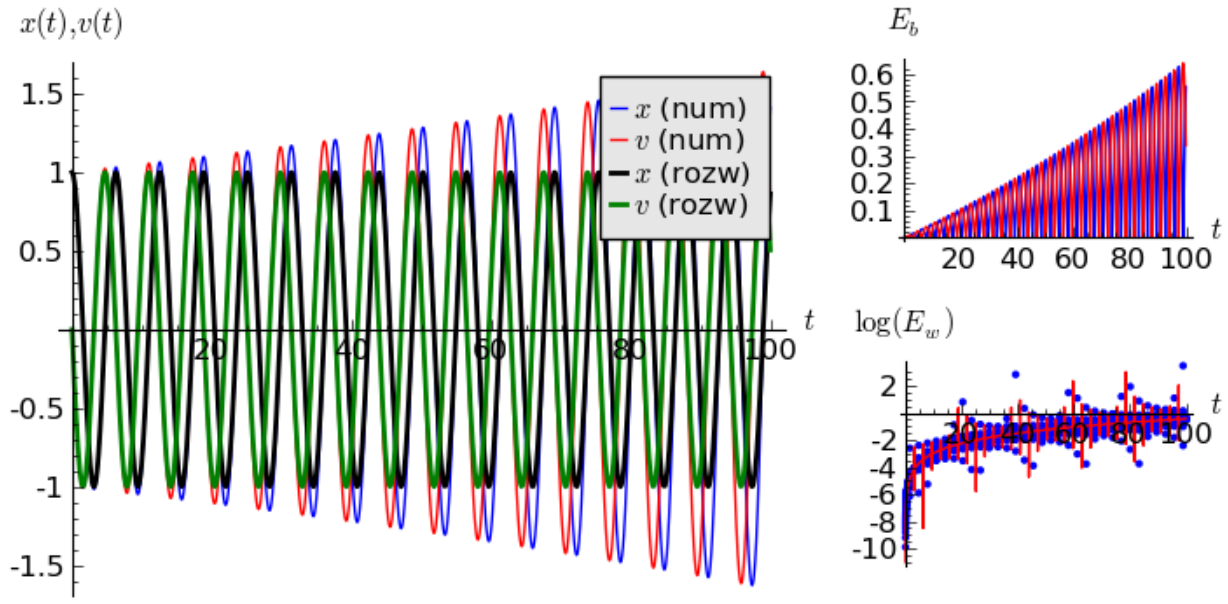
Po ustaleniu warunków początkowych $x(0) = x_0$ oraz $v(0) = v_0$ możemy rozpocząć normalną procedurę symulacji - wybieramy krok czasowy *h*, ustalamy parametry równania i do dzieła.

```
h = 0.01 # skok
N = 100 # liczba krokow

x0 = 1
v0 = 0
omega0 = 1

lista_x = [x0]
lista_v = [v0]
lista_t = [0]
for i in xrange(N):
  lista_x.append(lista_x[i] + lista_v[i] * h)
  lista_v.append(lista_v[i] - omega0^2 * lista_x[i] * h)
  lista_t.append(lista_t[i] + h)
```

Wykorzystamy też Sage do obliczenia rozwiązania analitycznego dla naszego zagadnienia.

```
var('t x omega x_0 v_0')
x = function('x', t)
assume(omega>0)
eq = diff(x, t, 2) + omega^2 * x == 0
solx = desolve(eq, x, ivar=t, ics=[0, x_0, v_0])
solv = diff(solx,t)
```

Teraz możemy zobaczyć jak dokładna jest metoda Eulera w przypadku równań wyższych rzędów. Poniżej znajdziecie wykres rozwiązań dla *h=0.01* i 10000 kroków.



Rysunek 8.4: Porównanie numerycznego ($x(t)$ linia niebieska, $v(t)$ linia czerwona) i analitycznego ($x(t)$ linia czarna, $v(t)$ linia zielona) rozwiązania zagadnienia oscylatora harmonicznego. Jak widać odchylenia od rozwiązań dokładnych zaczynją być znaczące już dla kilku kroków symulacji. Błąd bezwzględny widnieje na prawym górny panelu; błąd względny wykreślony jest na prawym dolnym panelu w skali logarytmicznej dla lepszej czytelności. Parametry użyte dla powyższej symulacji: $x_0 = 1, v_0 = 0, \omega_0 = 1, h = 0.01, N = 10000$.

Inną, aczkolwiek trudniejszą metodą będzie użycie algorytmów wyższego rzędu takich jak schemat Rungego-Kutty (2-giego, 4-tego i wyższych rzędów). Dociekliwy student może zajrzeć tutaj.

Zadanie D1.3 Przeprowadź podobną symulację dla innych wartości h. Wykreśl zależność błędów względnego i bezwzględnego w funkcji wartości h. Błędy badaj po rzeczywistym czasie symulacji $T_E = 100$.

Zadanie D1.4 Rozwiązać numerycznie równania

1. $\dot{x}(t) = -kx^3$
2. $\dot{x}(t) = F$
3. $m\ddot{x}(t) = mg$
4. $\ddot{r}(t) = 4\epsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right], r > 0$
5. $m\ddot{x}(t) = -\gamma\dot{x}(t) - kx(t)$
6. $m\ddot{x}(t) = -kx(t) - \gamma\dot{x}(t) + a\cos(\omega t)$

Dla każdego przypadku (A-F) należy

1. narysować *x* (dla D-F również *v*) w funkcji *t* (opisać osie),
2. odpowiedzieć na pytanie: z jakim ruchem mamy do czynienia [dla jakich parametrow równania ruch jest cykliczny (periodyczny), dla jakich rozwiązanie jest stałe (niezmienne w czasie)...],
3. znaleźć błąd względny i bezwzględny, wykreślić w funkcji czasu.

Pytania:

1. Czym różni się przypadek E od F?
2. Co opisuje potencjał w D? Jakie ma zastosowanie w fizyce?

Rozwiązania zadań

9.1 Rozdział 1

TBA

9.2 Rozdział 2

Zadanie 2.4.1 Należy rozwinąć w szereg funkcję $e^{\alpha x}$ i zauważyć, że

$$e^x + e^{-x} = 1 + x + \frac{x^2}{2} + \cdots + 1 - x + \frac{x^2}{2} + \cdots = 2 + x^2 + \cdots = 2 + \sum_{i=1}^{\infty} x^{2i} \geq 2$$

$$f(x) = \frac{1}{2 + e^x + e^{-x}} = \frac{1}{4 + \sum_{i=1}^{\infty} x^{2i}} \leq \frac{1}{4 + x^2}$$

Co daje nam ograniczenie poprzez rozkład Cauchy'ego dla $\sigma = 2$. Łatwo policzyć stosunek obu rozkładów i znaleźć $c = \pi/2$. Dalej należy postępować jak w przykładzie z wykładu.

9.3 Rozdział 3

Indices and tables

- *genindex*
- *modindex*
- *search*